
# Encentuate® Identity and Access Management (IAM)

## *Provisioning Integration Guide*

Product version 3.6

Document version 3.6.2

# Copyright notice



# Trademarks



# Contact information

For more information about this product or any support enquiries, contact us:

To log a support incident: https://customercare.encentuate.com

To reach us by phone:

- Americas: +1-800-ENCENTUATE ext 5 (+1-866-362-3688 ext 5)
- Asia Pacific: +65-6862-7085

Email: customercare@encentuate.com

# Table of Contents

# About This Guide

Welcome to the Encentuate IAM Provisioning Integration Guide.

Use this guide to configure, manage, and troubleshoot the different provisioning integration solutions for Encentuate IAM.

## Purpose

This guide provides procedures to help configure and maintain the provisioning integration solutions against Encentuate IAM.

## Audience

The target users for this integration guide are highly technical users that can understand how an Encentuate product can be enhanced and customized for user provisioning purposes.

## What's in this guide

About IAM Provisioning provides an overview of Encentuate IAM and how other provisioning solutions can integrate with IAM.

Encentuate Provisioning API describes the Encentuate IAM Provisioning API for provisioning and its integration mechanism with third-party identity provisioning systems.

IBM Tivoli Integration details setup and configuration steps required for the integration of the IBM Tivoli Identity Manager (ITIM) provisioning system with Encentuate IAM. It also provides procedures to create an AccessProfile for ITAM and to configure ITAM as an authentication service.

M-Tech ID-Synch defines setup and configuration steps required for the integration of the M-Tech ID-Synch provisioning system with Encentuate IAM.

Appendices provides additional information on troubleshooting and other ways of customizing and enhancing ITIM with Encentuate IAM

Glossary and Abbreviations defines all the commonly-used terms and abbreviations used throughout the guide.

# Document conventions

Refer to this section to understand the distinctions of formatted content in this guide.

## Main interface elements

The following are highlighted in bold text in the guide: dialog boxes, tabs, panels, fields, check boxes, radio buttons, fields, buttons, folder names, policy IDs/names, and keys. Examples are: **OK**, **Options** tab, and **Account Name** field.

## Navigation

All content that helps users navigate around an interface is italicized (for example: *Start >> Run >> All Programs*)

## Cross-references

Cross-references refer you to other topics in the guide that may provide additional information or reference. Cross-references are highlighted in green and display the referring topic's name (for example: Document conventions).

## Hyperlinks

Hyperlinks refer you to external documents or Web pages that may provide additional information or reference. Hyperlinks are highlighted in blue and display the actual location of the external document or Web page (for example: http://www.encentuate.com).

## Scripts, commands, and code

Scripts, commands, or code are those entered within the system itself for configuration or setup purposes, and are usually formatted in a Courier font.

For example:

```
<script language="JavaScript">

<!--

  ht_basename = "index.php";

  ht_dirbase = "";

  ht_dirpath = "/" + ht_dirbase;
```

```
//-->

</script>
```

### Tips or Hints

*Tips or hints help explain useful information that would help perform certain tasks better.*

### Warnings

*Warnings highlight critical information that would affect the main functionalities of the system or any data-related issues.*

# About IAM Provisioning

Provisioning systems have increasingly become critical components of the enterprise identity and access management strategy. A provisioning system provides identity lifecycle management for application users in enterprises and manages their credentials.

Encentuate IAM, an enterprise access security solution, provides real-time implementation of access security policies for users and applications. An integration between a provisioning system with Encentuate IAM access security solution results in a complete identity and access management solution.

The complete solution provides automatic application account provisioning, a central view of all application accounts, sign-on/sign-off automation, authentication management, user-centric audit logs and report generation, and centralized de-provisioning for all accounts.

This chapter covers the following topics:



## About Identity and Access Management (IAM)

Encentuate IAM is the first enterprise access security (EAS) solution that allows enterprises to simplify, strengthen, and track access to their digital assets and physical systems. Enterprises do not have to choose between strong security and convenience; they can have both.

Encentuate IAM combines sign-on/sign-off automation, authentication management and user tracking to provide a seamless path to strong digital identity. Encentuate IAM transparently increases security, enhances user convenience, and provides integrated access across existing information, network, and physical systems.

Encentuate IAM has two main components: Encentuate AccessAgent and Encentuate IMS Server. User credentials are stored in an Encentuate Wallet on the IMS Server. A user logs on to the Wallet by presenting one or more authentication factors to AccessAgent. AccessAgent then performs automatic sign-on to any enterprise application, which can be Windows-based, Web-based, mainframe-based or terminal-based.

# Provisioning key features and benefits

An identity provisioning system helps enterprises ensure that the right users have access to the right applications and infrastructure. It provides a secure, automated and policy-based identity lifecycle management solution that helps enterprises automate provisioning and de-provisioning of all user accounts, and provides a centralized view of all application credentials.

The embedded provisioning engine creates user credentials based on policies and defines the entitlements of the user accounts. It quickly connects users to appropriate enterprise resources while reducing administrative workload.

## Greater security through strict policy definition and enforcement

Encentuate IAM enforces strict, user-defined access security policies, and ensures all users meet these policies by managing their access privileges. Security is not compromised in any way, because Administrators do not see users' passwords – they only manage user access rights. This significantly strengthens the security culture within an enterprise. Encentuate IMS Connectors provide seamless enforcement of password change and fortification without user inconvenience.

## Enforcement of regulatory compliance requirements

Encentuate strengthens information access and provides user-centric audit logs, while provisioning systems enforce access control for sensitive data. Together, the Encentuate IAM's Provisioning API and the selected integration provide enterprises with a way to comply with regulations, such as Sarbanes-Oxley, the Gramm-Leach-Bliley Act, HIPAA, and the California SB 1386.

## Improved employee productivity

Encentuate IAM assists users in managing their credentials. When users launch an application, AccessAgent auto-fills their credentials and authenticates the users to the application server.

Integrating Encentuate IAM with another provisioning system provides convenience to users by consolidating credential management, allowing Administrators to centrally manage and administer users' consolidated accounts. Users do not have to remember any application passwords.

### Streamlined user-centric administration

The integration with Encentuate IAM eliminates the need for Administrators to manage too many application accounts. Administrators can automatically provision and administer users' multiple application accounts from one centralized system.

Integration with Encentuate IAM also eliminates the need to update users on changes to application credentials since application credential changes are automatically updated to the Wallets; all that users have to know is how to log on to the Wallet; any new application credential will automatically show up in the Wallet.

### Fast return on investment

The integration of Encentuate IAM's Provisioning API with another provisioning system allows Administrators to manage multiple user accounts from one location. The centralized process renders manual updates on individual accounts obsolete, which results in the significant reduction of time and costs associated with user account management. Consequently, the time and costs saved can be channeled to other areas of network administration.

# IAM integration solutions

Encentuate IAM can integrate with various identity provisioning systems to provide a complete identity and access management solution.

At present, the following provisioning systems can integrate with Encentuate IAM. For more information, see the separate chapters on each system.

- Encentuate Provisioning Application Programming Interface (API), see Encentuate Provisioning API
- IBM Tivoli, see IBM Tivoli Integration
- M-Tech ID-Synch, see M-Tech ID-Synch

# Encentuate Provisioning API

This chapter describes the Encentuate IAM Provisioning API for provisioning and its integration mechanism with third-party identity provisioning systems.

The chapter provides a reference for the SOAP API, as well as a guide for developing the SOAP client and configuring the IMS Server. For identity provisioning systems that support Java-based connectors, this chapter can also serve as a reference for the Java API, as well as a guide for developing the integration module and configuring the IMS Server.

This chapter covers the following topics:



# About Encentuate API

While the identity provisioning system provides the identity lifecycle management for application users, Encentuate IAM provides real-time implementation of access security policies for users and applications.

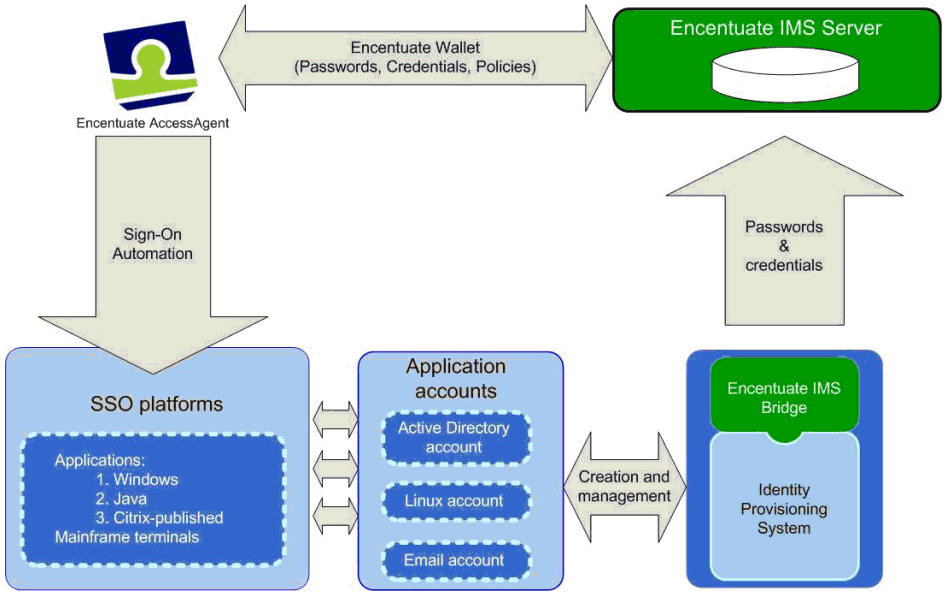
The integrated solution delivers seamless identity and access management that provides application account provisioning, a centralized view of all application accounts, sign-on/sign-off automation, authentication management, user-centric audit logs and reporting, and centralized de-provisioning of all accounts. The identity provisioning system needs to communicate with the IMS Server in order to populate and manage credentials in the Wallet.

The Encentuate IMS Bridge offers a Java interface to the identity provisioning system to communicate with the IMS Server. If it is not possible to use the Java interface, an Encentuate IMS Bridge that communicates directly with the IMS Server using SOAP will have to be developed for the identity provisioning system.

The workflow of the provisioning process is illustrated in the following diagram:


Encentuate integration overview

Communications between the identity provisioning system and the IMS Server are done using Simple Object Access Protocol (SOAP) over HTTPS. When the identity provisioning system provisions new users or new accounts for a user application, resets application passwords for users, de-provisions an enterprise user or an application account, it makes appropriate SOAP calls to the IMS Server with relevant account data information.




Communication process between the provisioning system and the IMS Server

The IMS Server accordingly creates a new user, populates the users' Wallet with the new account data, updates credentials, deletes accounts or revokes a user.

With this integration, users can enjoy single sign-on to provisioned accounts immediately upon sign on to Encentuate, and Administrators can revoke application access centrally by automatically removing credentials from the Encentuate Wallet.

# Minimum requirements

- Encentuate IMS Server: 3.0.0.0 and above
- Encentuate AccessAgent: 3.0.3.4 and above
- (Java API only) IMS Bridge should be deployed with Sun JVM version 1.4.x.

# Available API types

The developer may choose between two sets of APIs to integrate an identity provisioning system with IAM. These are described in detail in the sections that follow.

- Java API for Provisioning

  This is the recommended API to use for identity provisioning systems that support Java-based connectors for integration with third-party systems. The Java API provides a wrapper around the SOAP API so as to simplify its operations. For example, encryption of application passwords is performed by the provided IMS Bridge, and hence, is transparent to the developer.

- SOAP API for Provisioning

  If the Java API cannot be used, the developer may choose to use the SOAP API instead. The advantage of the SOAP API is that it is programming language independent, and hence, the provisioning agent can potentially be written in any programming language native to the identity provisioning system.

# Using Java API for provisioning

Encentuate provides a set of Java APIs for integration with identity provisioning systems.

The standard distribution of provisioning bridge contains the following directories:

- /bin: contains binary executables or scripts to invoke functions provided by IMS Bridge
- /lib: contains libraries for IMS Bridge
- /docs: contains configuration and deployment guide
- /config: sample configuration files for IMS Bridge

# Configuring a certificate store for the IMS Bridge (Java Provisioning API)

The IMS Bridge communicates with the IMS Server using one-way SSL. This means that the IMS Bridge needs to trust the IMS SSL certificate. If you are deploying the IMS Bridge on an application server, where there is already one common trust store shared by different applications, you need to import the IMS SSL certificate into the key store as one trusted CA entry.

Alternatively, you need to create one key store using the Java key tool utility. Then, you need to configure the IMS Bridge to use the above trusted store.

Be sure to complete the steps in Configuring The IMS Server.

# Configuring the IMS Bridge

The IMS Bridge is packaged with a sample configuration file as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

 <Config>

 <main>

   <ims.serverName>

   <value xml:lang="en">ims.yourcompany.com</value>

   </ims.serverName>

   <ims.httpsPort>

   <value xml:lang="en">443</value>

   </ims.httpsPort>

    <ims.httpPort>

   <value xml:lang="en">80</value>
```

```
    </ims.httpPort>

    <ims.servicePath>

    <value xml:lang="en">/ims/services</value>

    </ims.servicePath>

    <provisioningbridge.trustStore>

<value xml:lang="en">test\config\test_keystore</value>

    </provisioningbridge.trustStore>

    <provisioningbridge.jvm.environment.initializer>

<value
xml:lang="en">encentuate.bridges.provisioning.GenericJvmEnviron
mentInitializer</value>

    </provisioningbridge.jvm.environment.initializer>

    <provisioningbridge.trustStorePassword>

     <value xml:lang="en">password</value>

    </provisioningbridge.trustStorePassword>

    <provisioningbridge.authenticationService.mapping>

    <value xml:lang="en">ActiveDirectory:dir_ad</value>

    <value xml:lang="en">LotusNotes:dir_notes</value>

    </provisioningbridge.authenticationService.mapping>

  </main>

  </Config>
```

Refer to the following descriptions of XML parameters. Some of the parameters are optional.

*ims.serverName*
The DNS name of the IMS Server.

*ims.httpsPort (Optional)*
The port IMS Server listens to for HTTPS request. The default is **443**.

*ims.httpPort (Optional)*
The port IMS Server listens to for HTTP request. The default is **80**.

*ims.servicePath (Optional)*
The root path of IMS services. The default is **/ims/services/**. Note that the value should start with "/".

*provisioningbridge.trustStore (Mandatory for CLT only)*
The trust store used by IMS Bridge (Example: **C:\path\to\truststore**). This configuration does not take effect if there is already one system property set for **javax.net.ssl.trustStore**.

*provisioningbridge.trustStorePassword (Mandatory for CLT only)*
Password of the trust store used by IMS Bridge. This configuration does not take effect if there is already one system property set for **javax.net.ssl.trustStorePassword**.

*provisioningbridge.password.encryption.algorithm (Optional)*
The algorithm that encrypts the provisioned application passwords. The default algorithm is **RSA/NONE/PKCS1Padding**.

*provisioningbridge.password.encryption.transformation (Optional)*
The transformation ID that corresponds to the encryption algorithm. The default is **RSA/NONE/PKCS1Padding/2048/ProvisionKeypair**.

*provisioningbridge.authenticationService.mapping (Optional)*
The mapping of application IDs on the host provisioning system to IMS Server's representation. The format of each value of this configuration key should follow the format: **prov_system_app_ID:IMS_server_app_ID**.

For example, you have configured an authentication service for Active Directory in IMS Server called **dir_encentuate.com**. However, the internal representation for the same authentication service in your provisioning system is **ENCENTUATE**. You will then need to include the following configuration key:

```
<provisioningbridge.authenticationService.mapping>

<value xml:lang=:"en">ENCENTUATE:dir_encentuate.com</value>

</provisioningbridge.authenticationService.mapping>
```

*provisioningbridge.jvm.environment.initializer (Optional)*
Name of a class that implements the JvmEnvironmentInitializer interface, which sets up the JVM environment such as JAVA system properties before IMS Bridge starts to run. The default is **encentuate.bridges.provisioning.GenericEnvironmentInitializer**.

# Developing an integration module using the IMS Bridge

To integrate with the IMS Bridge after installing and configuring the IMS Bridge:

1. Develop against the IMS Bridge Java API and compile.
2. Place all necessary classes in the JRE class path.

The following sample code uses the IMS Bridge Java API. It creates and revokes a user with user name "james" in a deployment that does not have multiple Active Directory (AD) domains.

```
import encentuate.bridges.provisioning;

public class IntegrationModule {

int userStatus;

List appAccounts = new List();


// Instantiates the Provisioning Bridge

ProvisioningBridge bridge =

 new ProvisioningBridge("C:\imsBridgeConfig.xml");


// Login to the IMS Bridge using the shared secret.

bridge.login("provisioning_system", "xyz123 ");


// Creates an account on the IMS Server for this user if

// he doesn't already have an one.

bridge.createImsAccount("james", "abcabc");


// Get registration status of the user account

// on the IMS Server.

userStatus = bridge.getRegistrationStatus("james");


// Once the IMS account is created, the provisioning system

// can app application account data to the credential

// Wallet of this IMS account.


// Adds a "SQL DB" account with user name "james.h" to the

// user's credential Wallet.

bridge.createAppAccountData
```

```
("james", "SQL DB", "james.h", "abcdef");


// Get a list of accounts in the user's

// credential Wallet.

appAccounts = bridge.getUserAccounts("james");


// Deletes an application account data from the user's

// credential Wallet

bridge.deleteAppAccountData("james", "SQL DB", "james.h");


// Revokes the IMS user and invalidates his credential

// Wallet

bridge.revokeImsAccount("james");


// Logout from the IMS Bridge

bridge.logout();
```

# Using the Command Line Tool (CLT)

The IMS Bridge APIs can be invoked using CLTs. To use CLT, you need to use a utility called commons-launcher in the /bin folder.

The following XML parameters must be set for CLT use:

- provisioningbridge.trustStore
- provisioningbridge.trustStorePassword

For more information, see the descriptions of the parameters in Configuring the IMS Bridge.

To provision a new IMS user, you can issue a command in the command prompt as follows:

```
java -cp C:\provisioningbridge\bin LauncherBootstrap run \

--configFile \

C:\provisioningbridge\config\provisioningBridge.xml \
```

```
--loginId bridge --password password --task addImsUser \

--imsUserId test.encentuate.com\james --imsUserPassword \
password --userPrincipalName james@test.encentuate.com \

--samAccountName james --domainDnsName test.encentuate.com
```

To add a new application account to the Wallet of an IMS user, you can issue a command in the command prompt as follows:

```
java -cp C:\provisioningbridge\bin LauncherBootstrap run \

--configFile \

 C:\provisioningbridge\config\provisioningBridge.xml \

--loginId bridge --password password --task add --imsUserId \
test.encentuate.com\james --appId \

dir_alpha.test.encentuate.com --appUserId james --appPassword \

password
```

To update an application account that has been stored in the Wallet of an IMS user, you can issue a command in the command prompt as follows:

```
java -cp C:\provisioningbridge\bin LauncherBootstrap run \

--configFile \

C:\provisioningbridge\config\provisioningBridge.xml --loginId \

bridge --password password --task set --imsUserId \

test.encentuate.com\james --appId \

dir_alpha.test.encentuate.com --appUserId james --appPassword \

password
```

To delete an application account that is stored in the Wallet of an IMS user, you can issue a command in the command prompt as follows:

```
java -cp C:\provisioningbridge\bin LauncherBootstrap run \

--configFile \

C:\provisioningbridge\config\provisioningBridge.xml --loginId \

bridge --password password --task delete --imsUserId \

test.encentuate.com\james --appId \

dir_alpha.test.encentuate.com --appUserId james
```

To check a user's registration status, you can issue a command in the command prompt as follows:

```
java -cp C:\provisioningbridge\bin LauncherBootstrap run \

--configFile \

C:\provisioningbridge\config\provisioningBridge.xml --loginId \

bridge --password password --task \

registrationStatus --imsUserId test.encentuate.com\james
```

To get a user's Wallet contents, you can issue a command in the command prompt as follows:

```
java -cp C:\provisioningbridge\bin LauncherBootstrap run \

--configFile \

C:\provisioningbridge\config\provisioningBridge.xml --loginId \

bridge --password password --task walletAccountInfo \

--imsUserId test.encentuate.com\james
```

# Java API classes

The IMS Bridge interfaces with third party identity provisioning systems via the class: encentuate.bridges.provisioning.ProvisioningBridge

The API of this class is as follows:

## ProvisioningBridge (Constructor)

Constructor of the class.

```
public ProvisioningBridge(String configFile)

   throws ProvisioningBridgeException
```

*Parameters:*

configFile – The full path name of the IMS configuration file.

*Throws:*

ProvisioningBridgeException – If there are errors while reading the configuration file, this exception is thrown with an error code BAD_CONFIGURATION.

## login

Authenticates the IMS Bridge to the IMS Server using a preconfigured shared secret. This method must be called first before other methods in the API can be used.

```
public void login(String serverId, String serverPassword)

   throws ProvisioningBridgeException, IllegalArgumentException
```

*Parameters:*

*serverId*
The case-insensitive ID of the server on which the IMS Bridge is running. This is preconfigured in the IMS Server as part of the shared secret for the IMS Bridge to login to IMS Server.

*serverPassword*
The corresponding cases-sensitive password for this shared secret.

*Throws:*

ProvisioningBridgeException – If login fails, with these possible error codes:

*INVALID_LOGIN*
If the serverId and serverPassword pair doesn't match any preconfigured shared secret, or if the IP of this server doesn't match that preconfigured on the IMS Server.

*ACCESS_DENIED*
If the login account has no access to IMS provisioning services.

*REMOTE_EXCEPTION*
If there are exceptions when calling IMS SOAP services.

## logout

Logs out from the IMS Server at the end of the session.

```
public void logout()
```

*Parameters:*

None

*Throws:*

ProvisioningBridgeException – If logout fails, with these possible error codes:

*AUTHENTICATION_NEEDED*
If this method is called before the IMS Bridge has logged on to the IMS Server.

*IMS_SERVER_ERROR*
If there was an error on the server that prevented this call from succeeding.

## createImsAccount

This method creates an account on the IMS Server using the given user name and initial password. User attributes are optional, depending on whether the deployment has multiple Active Directory (AD) domains. An IMS account is necessary for all the other provisioning tasks.

```
public void createImsAccount(String imsUserId, String
initialPassword)

   throws ProvisioningBridgeException

public void createImsAccount(String imsUserId,String
initialPassword, Hashtable userAttrs)

   throws ProvisioningBridgeException
```

### *Parameters:*

*imsUserId*
The user name for the new account on the IMS Server. This should be the same as the user's primary enterprise identity, e.g., domain user name. In a deployment with multiple AD domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*initialPassword*
The initial password to be set on IMS Server for the new account.

*userAttrs*
A hash table that contains user attributes in name-value pairs. This parameter should be supplied if the deployment has multiple AD domains. Currently, the following user attributes are supported:

- userPrincipalName (optional): UPN of the user account. (This is only useful when enterprise directory is AD.)
- domainDnsName: DNS domain name of the enterprise directory.
- samAccountName: sAMAccountName of the AD user account. (This is only useful when enterprise directory is AD.)
- directoryId (optional): ID of enterprise directory that holds the user account. (This is optional if there is only one enterprise directory configured in the IMS Server.)

### *Throws:*

ProvisioningBridgeException – If a new account cannot be created on the IMS Server, with these possible error codes:

*AUTHENTICATION_NEEDED*
If this method is called before the IMS Bridge has logged on to the IMS Server.

*USER_IS_REGISTERED*
If imsUserId already exists on the IMS Server.

*USER_IS_REVOKED*
If imsUserId is registered but is already revoked.

## deleteImsAccount

This method deletes a user account from the IMS Server and invalidates the user's credential Wallet. An account of the same user name can be created subsequent to this call.

```
public void deleteImsAccount(String imsUserId)

   throws ProvisioningBridgeException
```

*Parameters:*

*imsUserId*
The user name of the account to be deleted from the IMS Server. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*Throws:*

ProvisioningBridgeException – If deletion fails, with these possible error codes:

*AUTHENTICATION_NEEDED*
If this method is called before the IMS Bridge has logged on to the IMS Server.

*USER_IS_REVOKED*
If imsUserId had been revoked on the IMS Server.

*REMOTE_EXCEPTION*
If invocation of IMS service fails.

## revokeImsAccount

This method revokes a user account from the IMS Server and invalidates the user's credential Wallet. An IMS user account cannot be created with the same user name as a revoked account.

```
public void revokeImsAccount(String imsUserId)

   throws ProvisioningBridgeException
```

*Parameters:*

*imsUserId*
The user name of the IMS account to revoke. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*Throws:*

ProvisioningBridgeException – If revocation fails, with these possible error codes:

*AUTHENTICATION_NEEDED*
If this method is called before the IMS Bridge has logged on to the IMS Server.

*REMOTE_EXCEPTION*
If invocation of IMS service fails.

## getRegistrationStatus

This method returns the registration status of a user account on the IMS Server.

```
public int getRegistrationStatus(String imsUserId)

   throws ProvisioningBridgeException
```

*Parameters:*

*imsUserId*
The user name of the IMS account to be checked. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*Returns:*

Returns one of the following result codes:

*ResultCode.USER_NOT_REGISTERED*
User is not found on the IMS Server.

*ResultCode.USER_IS_REGISTERED*
User is found on the IMS Server and is registered.

*ResultCode.USER_IS_REVOKED*
User is found on the IMS Server but is revoked.

*ResultCode.DATASTORE_EXCP*
A database access error has occurred.

*ResultCode.NOT_OK*
The method could not return the registration status due to some error.

*Throws:*

ProvisioningBridgeException – If the call fails, with these possible error codes:

*AUTHENTICATION_NEEDED*
If this method is called before the IMS Bridge has logged on to the IMS Server.

*IMS_SERVER_ERROR*
If there was an error on the server that prevented this call from succeeding.

## addAppAccountData

This method adds an application account data to the user's credential Wallet. The account must not already exist in the Wallet.

```
public void addAppAccountData(String imsUserId, String appId,
String appUserId,String appPassword)

   throws ProvisioningBridgeException
```

*Parameters:*

*imsUserId*
The user name of the IMS account whose credential Wallet is to be modified. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*appId*
The provisioning system's unique identifier for the application to which this account data belongs.

*appUserId*
The user name of the application account data to add.

*appPassword*
The password of the application account data to add.

*Throws:*

ProvisioningBridgeException – If the application account data cannot be set in the user's Wallet, with these possible error codes:

*AUTHENTICATION_NEEDED*
If this method is called before the IMS Bridge has logged on to the IMS Server.

*USER_IS_NOT_REGISTERED*
If imsUserId is not registered on the IMS Server.

*USER_IS_REVOKED*
If imsUserId had been revoked on the IMS Server.

*ACCOUNT_DATA_EXISTS*
If the account data already exists.

*UNKNOWN_AUTH_SERVICE*
If appId did not map to a valid application identifier on the IMS Server.

*IMS_SERVER_ERROR*
If there was an error on the server that prevented this call from succeeding.

## updateAppAccountData

This method updates an application account data on the user's credential Wallet. The account must already exist in the Wallet.

```
public void updateAppAccountData(String imsUserId, String
appId, String appUserId, String appPassword)

    throws ProvisioningBridgeException
```

*Parameters:*

*imsUserId*
The user name of the IMS account whose credential Wallet is to be modified. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*appId*
The provisioning system's unique identifier for the application to which this account data belongs.

*appUserId*
The user name of the application account data to update.

*appPassword*
The password of the application account data to update.

*Throws:*

ProvisioningBridgeException – If the application account data cannot be set in the user's Wallet, with these possible error codes:

*AUTHENTICATION_NEEDED*
If this method is called before the IMS Bridge has logged on to the IMS Server.

*USER_IS_NOT_REGISTERED*
If imsUserId is not registered on the IMS Server.

*USER_IS_REVOKED*
If imsUserId had been revoked on the IMS Server.

*ACCOUNT_DATA_DOESNT_EXIST*
If the account data doesn't already exist.

*UNKNOWN_AUTH_SERVICE*
If appId did not map to a valid application identifier on the IMS Server.

*IMS_SERVER_ERROR*
If there was an error on the server that prevented this call from succeeding.

## deleteAppAccountData

This method deletes an application account data from the user's credential Wallet on the IMS Server.

```
public void deleteAppAccountData(String imsUserId, String
appId, String appUserId)

   throws ProvisioningBridgeException
```

*Parameters:*

*imsUserId*
The user name of the IMS account whose credential Wallet is to be modified. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*appId*
The provisioning system's unique identifier for the application to which this account data belongs.

*appUserId*
The user name of the application account data to delete.

*Throws:*

ProvisioningBridgeException – If the application account cannot be deleted from the user's Wallet, with these possible error codes:

*AUTHENTICATION_NEEDED*
If this method is called before the IMS Bridge has logged on to the IMS Server.

*USER_IS_NOT_REGISTERED*
If imsUserId is not registered on the IMS Server.

*USER_IS_REVOKED*
If imsUserId had been revoked on the IMS Server.

*UNKNOWN_AUTH_SERVICE*
If appId did not map to a valid application identifier on the IMS Server.

*IMS_SERVER_ERROR*
If there was an error on the server that prevented this call from succeeding.

### getUserAccounts

This method returns a list of accounts in the user's credential Wallet. Only password accounts and accounts that are provisioned would be returned.

```
public List getUserAccounts(String imsUserId)

   throws ProvisioningBridgeException
```

*Parameters:*

*imsUserId*
The user name of the IMS account to be checked. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*Returns:*

Returns a list of accounts. If the user does not have any accounts, the list is empty. Otherwise, each entry in the list is an instance of Map object. The Map object contains selected information for one account and the following keys are available:

*USER_ID*
User name of the account.

*AUTHSVC_ID*
The authentication service ID of the account.

*Throws:*

ProvisioningBridgeException – If the call fails, with these possible error codes:

*AUTHENTICATION_NEEDED*
If this method is called before the IMS Bridge has logged on to the IMS Server.

*IMS_SERVER_ERROR*
If there was an error on the server that prevented this call from succeeding.

# Using SOAP API for provisioning

SOAP (Simple Object Access Protocol) is a protocol for exchanging XML-based messages over a computer network, normally using HTTP. SOAP forms the foundation layer of the Web services stack, providing a basic messaging framework that more abstract layers can build on.

SOAP services are defined using WSDL (Web Services Definition Language) and are accessible via a URL which is known as a SOAP endpoint.

Encentuate IAM provides a SOAP API for identity provisioning systems to communicate with the IMS Server to perform provisioning. With the SOAP API, the request interface is an object in your application's native programming language.

A third-party SOAP client can be used to generate business-object interfaces and network stubs from a WSDL document that specifies the IMS Server message schema, the service address, and other information.

The SOAP client handles the details of building the SOAP request and sending it to the IMS Server, whereas your application would work with data in the form of object properties, and it sends and receives the data by calling object methods.

Identity provisioning for IAM requires the use of two sets of SOAP APIs:

- **API for server authentication**

  For the provisioning agent to log on and log off the IMS Server. Provisioning agents must log on to the IMS Server before invoking other API operations.

- **API for provisioning service**

  For creating/deleting/revoking Encentuate users, and adding/setting/removing application account credentials.

A typical identity provisioning system contains provisioning agents for provisioning users and applications on third-party systems. It is assumed henceforth that the provisioning agent would be using the SOAP API to integrate with the IMS Server.

The provisioning agent first sets up an IMS Server session by logging on to the IMS Server. It provisions Encentuate users by specifying their user names and initial passwords. It can also provision application credentials by specifying the application user names and passwords.

When necessary, the provisioning agent can also call the appropriate operations to reset application passwords, remove application credentials, and delete or revoke Encentuate users. Finally, the provisioning agent terminates the session by logging off the IMS Server.

For more information on integrating an identity provisioning system with the IMS Server using the SOAP API, see Developing the SOAP client and Configuring the IMS Server.

*Usage of the API for provisioning a typical user is as follows:*

1. Call loginByPassword to log on to the IMS Server.
2. Call preProvisionImsUser to provision an Encentuate user.
3. Call createWallet to create a Wallet for the Encentuate user.
4. Call getProvisioningCert to obtain the user's provisioning certificate that would be used to encrypt application passwords.

5. Call addAccountCredential for each application to be provisioned for the user.

6. Call terminateSession to log off the IMS Server.

*Usage of the API for resetting an application password for a user is as follows:*

1. Call loginByPassword to log on to the IMS Server.

2. Call getProvisioningCert to obtain the user's provisioning certificate that would be used to encrypt application passwords.

3. Call setAccountCredential for the application.

4. Call terminateSession to log off the IMS Server.

*Usage of the API for removing application account credentials for a user is as follows:*

1. Call loginByPassword to log on to the IMS Server.

2. Call removeAccountCredential for the application.

3. Call terminateSession to log off the IMS Server.

*Usage of the API for deleting or revoking an Encentuate user is as follows:*

1. Call loginByPassword to log on to the IMS Server.

2. Call deleteImsAccount or revokeImsAccount to delete or revoke the Encentuate user.

3. Call terminateSession to log off the IMS Server.

*Usage of the API for checking the status of an Encentuate user and obtaining the list of accounts in the credential Wallet is as follows:*

1. Call loginByPassword to log on to the IMS Server.

2. Call getRegistrationStatus to check whether the user is registered. If so, call getUserAccounts to obtain the list of accounts in the credential Wallet.

3. Call terminateSession to log off the IMS Server.

# Developing the SOAP client

The provisioning agent (Encentuate IMS Bridge) must be developed as a SOAP client.

The SOAP API has been tested on the following platforms:

- Microsoft .NET with Visual Studio .NET
- Apache Axis

Other SOAP client environments can be used as long as they support standard SOAP messages.

SOAP tools consume WSDL to generate SOAP client code. The WSDL for this API can be obtained from an installed IMS Server at the following URLs (**imsserver** should be replaced by the hostname of your IMS Server):

- https://imsserver/ims/services/encentuate.ims.service.ProvisioningService?wsdl
- https://imsserver/ims/services/encentuate.ims.service.ServerAuthentication?wsdl

You can also refer to WSDL for Server Authentication and WSDL for Provisioning Service for the list of WSDLs.

A sample code is available in the following folder that accompanies this document: Windows application using Visual C# (win_cs folder).

*To develop the SOAP client (using Visual Studio .NET):*

1. Add a Web Reference, directing it to the WSDL for Server Authentication. Name it **ImsServerAuthentication**.
2. Add a Web Reference, directing it to the WSDL for Provisioning Service. Name it **ImsProvisioningService**.
3. In the appropriate code, create a new **ImsServerAuthentication.ServerAuthenticationService** object.
4. In the appropriate code, create a new **ImsProvisioningService.ProvisioningServiceService** object.
5. Where needed, call the objects' methods.

*To develop the SOAP client (using Apache Axis):*

1. Use WSDL2Java to automatically create Java stubs and classes.
2. In the appropriate code, create a new **EncentuateServerAuthenticationServerAuthenticationSoapBindingStub** object, directing it to the WSDL for ServerAuthentication.
3. In the appropriate code, create a new **EncentuateProvisioningServiceProvisioningServiceSoapBindingStub** object, directing it to the WSDL for ProvisioningService.

❹ Where needed, call the objects' methods.

# Configuring the IMS Server

To configure the IMS Server for SOAP API, refer to the procedure in Configuring The IMS Server. However, note the following:

- The provisioning agent communicates with the IMS Server using one-way SSL. This means that the provisioning agent needs to trust the IMS Server's SSL certificate.

  If you are deploying the provisioning agent on an application server, where there is already a common trust store shared by different applications, import the IMS Server's SSL certificate into the key store as one trusted certification authority entry.

  On the Java platform, you can create a key store using the Java key tool utility. On the Visual Studio .NET platform, you can store the IMS Server's SSL certificate in the Windows certificate store using the Certificates snap-in.

  You can use Internet Explorer to download the IMS Server's SSL certificate into the Windows certificate store by visiting https://imsserver/ where **imsserver** is the IMS Server's hostname, and then click **View Certificate** and proceed to install the certificate in the Local Computer certificate store.

  *Go to Install Certificate >> Next >> Place all certificates in the following store >> Browse >> Show physical stores >> Trusted Root Certification Authorities >> Local Computer >> OK >> Next >> Finish*.

- Ensure that the SOAP client (provisioning agent) specifies the correct provisioning agent name, shared secret, authentication service IDs, and account types when using the API.

# SOAP API data types

This section specifies the data types and values that are used by this API.

## resultCode

```
<element name="resultCode" type="xsd:int" />
```

*Synopsis:*

The result code that is returned by an operation to indicate the operation's success or the reason for failure.

*Value:*

| Identifier | Value (Hex) | Description |
|---|---|---|
| OK | 0x00000000 | Success. |
| NOT_OK | 0x50000001 | Generic failure. |
| BAD_CONFIGURATION | 0x13000110 | Bad IMS Server configuration. |
| ACCESS_DENIED | 0x53000220 | Incorrect shared secret or SOAP client IP address is not allowed. |
| DATASTORE_EXCP | 0x23005000 | A database error has occurred. |
| BAD_INPUT_PARAMETERS | 0x53000320 | An input parameter is null or empty. |
| INVALID_LOGIN_CLIENT_IP_MISSING | 0x53000254 | Unable to log on because SOAP client IP address could not be found. |
| INVALID_SESSION | 0x53000200 | Session ID is not valid. |
| UNKNOWN_AUTH_SERVICE | 0x53008101 | The requested authentication service does not exist. |
| INVALID_AUTH_SERVICE | 0x53008150 | The requested authentication service has not been correctly defined. |
| INVALID_ACCOUNT_DATA_TEMPLATE | 0x53008151 | The account data template specified for the authentication service does not match the fields given as inputs. |
| USER_IS_REGISTERED | 0x53000282 | The specified user is registered. |
| USER_IS_REVOKED | 0x53000259 | The specified user has been revoked. |
| USER_NOT_REGISTERED | 0x53000284 | The specified user is not yet registered. |
| UNSUPPORTED_AUTH_MECHANISM | 0x53008251 | The authentication service does not support authentication by password. |

| Identifier | Value (Hex) | Description |
|---|---|---|
| UNKNOWN_DATA_STORAGE_TEMPLATE | 0x53008301 | There is an error obtaining the storage template for the authentication service. |
| ACCOUNT_ALREADY_EXISTS | 0x53008402 | The account being added already exists for the user. |
| ENTRY_NOT_FOUND | 0x23005530 | The specified user is not found. |
| UNEXPECTED_WARNING | 0x23000002 | An encryption error has occurred. |
| UNKNOWN_ACCOUNT_DATA_TEMPLATE | 0x53008102 | There is an error obtaining the account data template for the authentication service. |
| ACCOUNT_NOT_FOUND | 0x53008403 | The account to be deleted does not exist. |

## resultString

```
<element name="resultString" nillable="true" type="xsd:string"
/>
```

*Synopsis:*

The result string that is returned by an operation to indicate the operation's output.

*Value:*

Depends on the operation.

## ResultMessage

```
<complexType name="ResultMessage">

 <sequence>

 <element name="resultCode" type="xsd:int" />

 <element name="resultString" nillable="true" type="xsd:string"
/>

 </sequence>

</complexType>
```

```
<element name="ResultMessage" nillable="true"
type="tns1:ResultMessage" />
```

*Synopsis:*

The composite result message that is returned by an operation to indicate the operation's success/failure as well as its output.

*Value:*

Depends on the operation.

## NameValue

```
<complexType name="NameValue">

 <sequence>

<element name="name" nillable="true" type="xsd:string"/>

<element name="value" nillable="true" type="xsd:string"/>

 </sequence>

</complexType>
```

*Synopsis:*

Name value pair for a user attribute.

*Value:*

For example, to set a user's mobile phone number for receiving Mobile ActiveCodes, name should be set to "gsmNumber" and value should be set to a mobile phone number of the format "1-617-12345678".

## ResultArrayMap

```
<complexType name="ResultArrayMap">

 <sequence>

 <element name="maps" nillable="true"
type="impl:ArrayOf_apachesoap_Map"/>

 <element name="resultCode" type="xsd:int"/>

 </sequence>

</complexType>

<element name="ResultArrayMap" nillable="true"
type="tns1:ResultArrayMap"/>
```

```
<complexType name="ArrayOf_apachesoap_Map">

 <complexContent>

 <restriction base="soapenc:Array">

 <attribute ref="soapenc:arrayType"
wsdl:arrayType="apachesoap:Map[]"/>

 </restriction>

 </complexContent>

</complexType>


<complexType name="Map">

 <sequence>

 <element maxOccurs="unbounded" minOccurs="0" name="item">

 <complexType>

 <all>

 <element name="key" type="xsd:anyType"/>

 <element name="value" type="xsd:anyType"/>

 </all>

 </complexType>

 </element>

 </sequence>

</complexType>
```

*Synopsis:*

The composite result message that is returned by an operation to indicate the operation's success/failure as well as an array of Maps as its output.

*Value:*

Depends on the operation.

# SOAP API operations

## Server authentication

This section specifies the operations that are offered by the API for server authentication.

### loginByPassword

```
public ResultMessage loginByPassword(

 String serverId, String password);
```

*Synopsis:*

To log on to the IMS Server as a provisioning agent.

This operation requests the IMS Server to issue a session ID that can be used in other provisioning operations.

*Arguments:*

*serverId*
The "name" of the provisioning agent. This should correspond to the provisioning agent name setting in the IMS Server configuration file (see IMS Server Configuration section).

*password*
The shared secret between the provisioning agent and the IMS Server.

*Return Value:*

Returns ResultMessage, which consists of resultCode and resultString.

Returns session ID in resultString if resultCode is OK.

Returns error message in resultString if resultCode is not OK.

*Remarks:*

This operation should be used to log on to the IMS Server before other operations of this API can be invoked. The session ID returned by this operation is to be used in other operations of this API.

### terminateSession

```
public int terminateSession(String sessionKey);
```

*Synopsis:*

To terminate the session.

This operation requests the IMS Server to terminate the session for the specified sessionId.

*Arguments:*

*sessionKey*
The session ID of the session to be terminated.

*Return Value:*

Returns ResultCode.

*Remarks:*

This operation should be used to terminate the session after all required user provisioning operations have been performed.

# Provisioning service

This section specifies the operations that are offered by the API for provisioning service. Before using any of these operations, a session ID must already be obtained through the loginByPassword operation of the API for server authentication.

## addAccountCredential

```
public int addAccountCredential(

 String sessionId, String enterpriseId, String authId,

 String accountType, String username, String password);
```

*Synopsis:*

To add provisioned account credentials for a user.

This operation requests the IMS Server to add a set of application account credentials for an existing Encentuate user. For security, the password is to be encrypted using the user's provisioning certificate.

*Arguments:*

*sessionId*
The session ID of the provisioning agent.

*enterpriseId*
The enterprise ID (Encentuate user name) of the user to be provisioned. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*authId*
The authentication service ID of the user account to be provisioned.

*accountType*
The type of account to be provisioned: “Password”, “OTP”, “MAC”, or “Certificate”.

*username*
The application user name of the user account to be provisioned.

*password*
The password in Base64, encrypted by the user’s provisioning encryption certificate in an XML snippet of the following form:

```
<password transformation="RSA/None/PKCS1Padding/2048/
ProvisionKeypair">

QtOpkL0eWD+7vMPLUIVYHJFHijY+2ggvm1C26raOXMZBURrbQRbgvXeI4SA5tuh
7EuBkLJWjC/fhivpBqmz2NmlersSUFZ4IQxYe0EXDtxBmkSF149I/
eieUVzhVcyvrzkP276FPX4YO1Miz/S4fq9o4Xs7W1r33Nu2tKSCVwvNWZ1R2/
DtRTxmHI5ibOR0Vs3ie7rdGpG75xY61gwwMUCFeF7VoFZT1TO7AXA7yT1AbOiE6
OiYHxh12VTASNPo8SegG1vZqjrxrzIUbilDloV620C6RhV7D74liXykhZmmxBH/
UWvaK3GlI1xE/Cva39hIEO1Uw8m1SPNi1gLqLKw==

</password>
```

*The password should be encoded in UTF-16LE encoding before it is encrypted. After encryption, the byte stream should be in Little-Endian order before conversion into Base64 encoding.*

*Return Value:*

Returns resultCode.

*Remarks:*

Before using this operation, the user’s provisioning certificate should already be obtained through the getProvisioningCert operation.

## createWallet

```
public ResultMessage createWallet(

 String sessionId, String enterpriseId, String walletId);
```

*Synopsis:*

To create a Wallet with the given enterprise ID.

This operation requests the IMS Server to create a Wallet for the provisioned user. A Wallet will only be created for the user if he does not already have a Wallet.

*Arguments:*

*sessionId*
The session ID of the provisioning agent.

*enterpriseId*
The enterprise ID (Encentuate user name) of the user to be provisioned. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*walletId*
The ID of the Wallet to be created. If this parameter is null or empty, a Wallet ID will be generated.

*Return Value:*

Returns ResultMessage, which consists of resultCode and resultString.

Returns Wallet ID in resultString if resultCode is OK.

*Remarks:*

This operation should be used to create a Wallet after a user has been provisioned.

## deleteImsAccount

```
public int deleteImsAccount(

 String sessionId, String enterpriseId);
```

*Synopsis:*

To delete an Encentuate user.

This operation requests the IMS Server to delete an Encentuate user and the corresponding Wallet. If deleted, a user will be completely removed from the IMS database. Another user of the same enterprise ID can be provisioned again later on.

*Arguments:*

*sessionId*
The session ID of the provisioning agent.

*enterpriseId*
The enterprise ID (Encentuate user name) of the user to be provisioned. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*Return Value:*

Returns resultCode.

## getProvisioningCert

```
public ResultMessage getProvisioningCert(

 String sessionId, String enterpriseId);
```

*Synopsis:*

To get the specified user's X.509 provisioning certificate.

This operation requests the IMS Server to return the X.509 certificate that can be used to encrypt passwords for provisioned account credentials.

*Arguments:*

*sessionId*
The session ID of the provisioning agent.

*enterpriseId*
The enterprise ID (Encentuate user name) of the user to be provisioned. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*Return Value:*

Returns ResultMessage, which consists of resultCode and resultString.

Returns user's provisioning encryption certificate in resultString if resultCode is OK. The certificate is a Base64-encoded X.509 (.CER) certificate.

*Remarks:*

This operation should be used to obtain a user's provisioning certificate before operations for setting account credentials are used: addAccountCredential and setAccountCredential.

## getRegistrationStatus

```
public int getRegistrationStatus(

 String sessionId, String enterpriseId);
```

*Synopsis:*

To get the registration status of a user account on the IMS Server.

This operation requests the IMS Server to return the registration status of a user account, which may not be found or may be revoked.

*Arguments:*

*sessionId*
The session ID of the provisioning agent.

*enterpriseId*
The enterprise ID (Encentuate user name) of the user to be provisioned. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*Return Value:*

Returns resultCode.

## getUserAccounts

```
public ResultArrayMap getUserAccounts(

 String sessionId, String enterpriseId);
```

*Synopsis:*

To get the list of accounts in a user's credential Wallet.

This operation requests the IMS Server to return the list of accounts in a user's credential Wallet. Only password accounts and accounts that are provisioned would be returned.

*Arguments:*

*sessionId*
The session ID of the provisioning agent.

*enterpriseId*
The enterprise ID (Encentuate user name) of the user to be provisioned. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*Return Value:*

Returns ResultArrayMaps, which consists of resultCode and an array of Maps. If the user does not have any accounts, the array is empty. Otherwise, each entry in the array is an instance of Map object. The Map object contains selected information for one account and the following keys are available:

*USER_ID*
User name of the account.

*AUTHSVC_ID*
The authentication service ID of the account.

## preProvisionImsUser

```
public int preProvisionImsUser(

 String sessionId, String enterpriseId, String initialPassword,

 NameValue[] attributes);
```

### *Synopsis:*

To create an Encentuate user.

This operation requests the IMS Server to create an Encentuate user with the specified enterprise ID (Encentuate user name) and initial Encentuate password.

### *Arguments:*

*sessionId*
The session ID of the provisioning agent.

*enterpriseId*
The enterprise ID (Encentuate user name) of the user to be provisioned. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*initialPassword*
The initial Encentuate password for the user.

*attributes*
Any IMS attributes (name value pairs) that are to be added for the new Encentuate user. null is an acceptable input. The following user attributes are to be used for a deployment with multiple AD domains:

- userPrincipalName (optional): UPN of the user account. (This is only useful when enterprise directory is AD.)
- domainDnsName: DNS domain name of the enterprise directory.
- samAccountName: sAMAccountName of the AD user account. (This is only useful when enterprise directory is AD.)
- directoryId (optional): ID of enterprise directory that holds the user account. (This is optional if there is only one enterprise directory configured in the IMS Server.)

### *Return Value:*

Returns resultCode.

## removeAccountCredential

```
public int removeAccountCredential(

 String sessionId, String enterpriseId, String authId,

 String accountType, String username);
```

*Synopsis:*

To remove provisioned account credentials for a user.

This operation requests the IMS Server to remove the specified account credentials for a user. The account must already exist for the user.

*Arguments:*

*sessionId*
The session ID of the provisioning agent.

*enterpriseId*
he enterprise ID (Encentuate user name) of the user to be provisioned. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*authId*
The authentication service ID of the user account to be provisioned.

*accountType*
The type of account to be provisioned: "Password", "OTP", "MAC", or "Certificate".

*username*
The application user name of the user account to be removed.

*Return Value:*

Returns resultCode.

## revokeImsAccount

```
public int revokeImsAccount(

 String sessionId, String enterpriseId);
```

*Synopsis:*

To revoke an Encentuate user.

This operation requests the IMS Server to revoke an Encentuate user and the authentication factors. If revoked, a user will be marked as revoked in the IMS database. New users are not allowed to re-use the enterprise ID of the revoked user.

*Arguments:*

*sessionId*
The session ID of the provisioning agent.

*enterpriseId*
The enterprise ID (Encentuate user name) of the user to be provisioned. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*Return Value:*

Returns resultCode.

## setAccountCredential

```
public int setAccountCredential(

 String sessionId, String enterpriseId, String authId,

 String accountType, String username, String password);
```

*Synopsis:*

To set provisioned account credentials for a user.

This operation requests the IMS Server to set the application password of an existing user account. This operation will fail if the account data does not already exist or the account is not an account of "Password" type.

*Arguments:*

*sessionId*
The session ID of the provisioning agent.

*enterpriseId*
The enterprise ID (Encentuate user name) of the user. In a deployment with multiple Active Directory (AD) domains, the format of the user name should be of the form dnsDomain\sAMAccountName (e.g., "encentuate.com\username").

*authId*
The authentication service ID of the user account to be set.

*accountType*
The type of account to be set: "Password", "OTP", "MAC", or "Certificate".

*username*
The application user name of the user account to be set.

*password*
The password in Base64, encrypted by the user's provisioning encryption certificate in an XML snippet of the following form:

```
<password

transformation="RSA/None/PKCS1Padding/2048/ProvisionKeypair">

QtOpkL0eWD+7vMPLUIVYHJFHijY+2ggvm1C26raOXMZBURrbQRbgvXeI4SA5tuh
7EuBkLJWjC/fhivpBqmz2NmlersSUFZ4IQxYe0EXDtxBmkSF149I/
eieUVzhVcyvrzkP276FPX4YO1Miz/S4fq9o4Xs7W1r33Nu2tKSCVwvNWZ1R2/
DtRTxmHI5ibOR0Vs3ie7rdGpG75xY61gwwMUCFeF7VoFZT1TO7AXA7yT1AbOiE6
OiYHxh12VTASNPo8SegG1vZqjrxrzIUbilDloV620C6RhV7D74liXykhZmmxBH/
UWvaK3GlI1xE/Cva39hIEO1Uw8m1SPNi1gLqLKw==

</password>
```

*The password should be encoded in UTF-16LE encoding before it is encrypted. After encryption, the byte stream should be in Little-Endian order before conversion into Base64 encoding.*

*Return Value:*

Returns resultCode.

*Remarks:*

Before using this operation, the user's provisioning certificate should already be obtained through the getProvisioningCert operation.

# Provisioning API setup and maintenance

*To provision a new user (Provisioning API):*

1. The provisioning system provides the Encentuate user name and initial Encentuate password to the IMS Server, which in turn will provision the Encentuate user on the IMS Server.

2. The user's Wallet will also be initialized on the IMS Server at this time so that application account credentials can then be added to the Wallet.

*Encentuate user accounts should be created before other application accounts. Application account credentials cannot be added to the user's Wallet before the user is provisioned on the IMS Server.*

❸ The users log on with their Encentuate user names and initial Encentuate passwords when they use AccessAgent for the first time. They will be prompted to change the initial password and the Wallet containing the provisioned account credentials will be downloaded from the IMS Server.

*To add an application account (Provisioning API):*

❶ The provisioning system provides the Encentuate user name and application credentials to the IMS Server, which in turn will add the application and other credentials to the user's Wallet.

❷ The next time the user logs on to AccessAgent, the AccessAgent will have the necessary credentials in the Wallet to automate sign-on to the new application. Applications can therefore be added without having to inform users of the new credentials. Users just need to sign-on to AccessAgent.

*To reset an application password (Provisioning API):*

❶ The provisioning system provides the Encentuate user name and the new application password to the IMS Server, which in turn updates the application password in the user's Wallet.

❷ The next time the user logs on to AccessAgent, the AccessAgent will have the updated application passwords in the Wallet to automate sign-on to the applications. Administrators can reset application passwords directly notifying each user. Users just need to sign-on to Encentuate to log on to the application.

*To delete an application account (Provisioning API):*

❶ The provisioning system provides the Encentuate user name and the application account name to the IMS Server, which in turn deletes the application account from the user's Wallet.

❷ The next time the user logs on to AccessAgent, AccessAgent will no longer have access to the deleted application's credentials in the Wallets, and cannot sign-on to the application on the users' behalf. Applications can be removed centrally and all access can be terminated automatically.

*To de-provision users (Provisioning API):*

❶ The provisioning system provides the Encentuate user name to be de-provisioned to the IMS Server, which will revoke the Encentuate user. The revocation of the Encentuate user will invalidate both the user's accounts and the user's Wallet on the server.

❷ If the user attempts to log on using AccessAgent, the log on will fail and Wallets that have been cached locally by AccessAgent will be revoked.

# IBM Tivoli Integration

The integration of Encentuate IAM with IBM Tivoli Identity Manager provides an immediate solution. As ITIM provides identity lifecycle management, Encentuate provides the real-time enforcement of strong identities by simplifying, strengthening, and tracking access to all applications.

This chapter details setup and configuration steps required for the integration of the IBM Tivoli Identity Manager (ITIM) provisioning system with the Encentuate IAM access security solution. An integrated workflow and the benefits of an integrated provisioning and access security solution is also discussed in this section. It is assumed that both ITIM and IMS Server have been installed.

This chapter covers the following topics:



# About IBM Tivoli integration

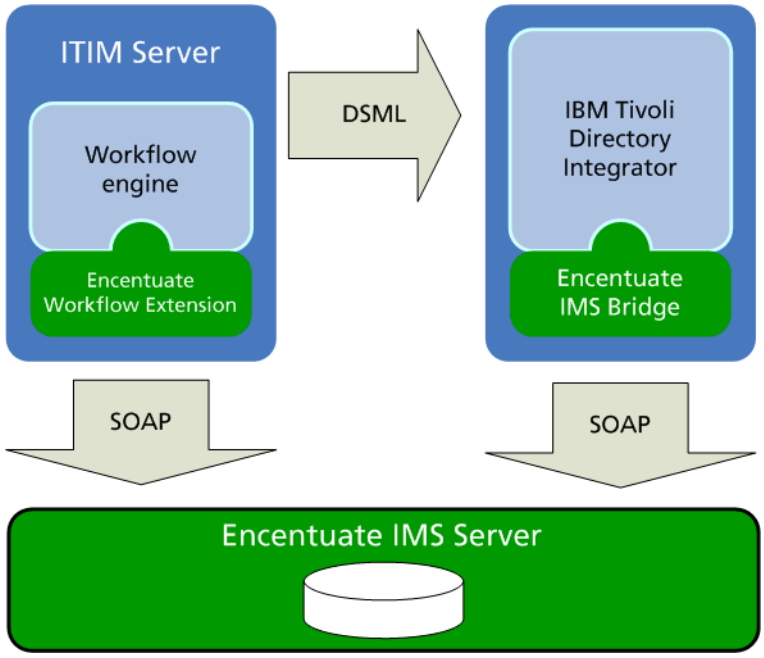
Encentuate IAM integrates with both Tivoli Access Manager (ITAM) and Tivoli Identity Manager (ITIM) to provide a complete identity and access management solution.

While ITIM provides the identity lifecycle management for application users, Encentuate IAM provides real time implementation of access security policies for users and applications.

The integrated solution delivers seamless identity and access management that provides application account provisioning, a centralized view of all application accounts, sign-on/sign-off automation, authentication management, user-centric audit logs and reporting, and centralized de-provisioning of all accounts.

Encentuate IAM is already integrated with ITIM, making it possible for users created in ITIM to be automatically provisioned in IAM. Application accounts that are provisioned using ITIM are also automatically populated in the corresponding users' Encentuate Wallets.

The workflow of the provisioning process is illustrated in the following diagram:




Encentuate integration overview (IBM Tivoli)

The IBM Tivoli Identity Manager needs to communicate with the IMS Server to populate and manage credentials in the Wallet. The Encentuate IMS Bridge and the Encentuate Workflow Extension are the interface engines that act as intermediaries between the IMS Server and the IBM Tivoli Identity Manager.

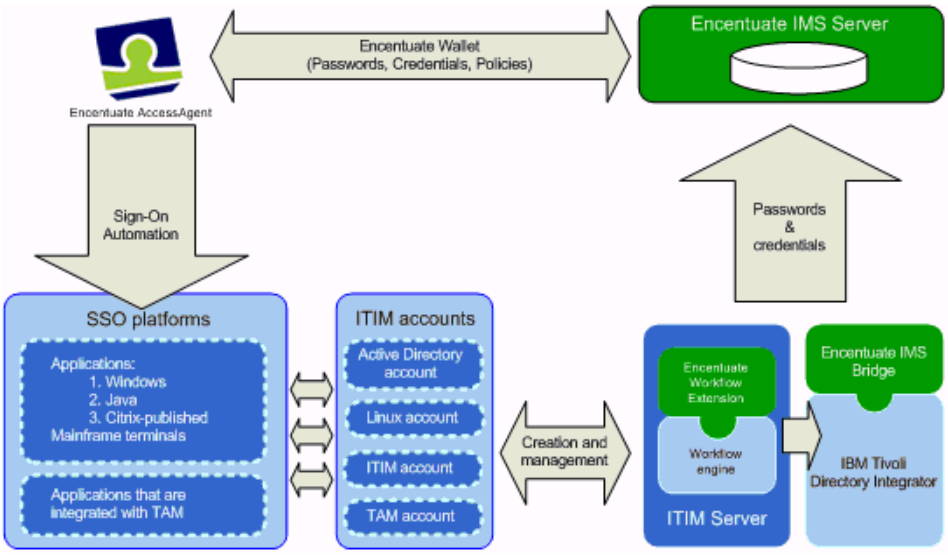
ITIM will connect to the IMS Server via the Encentuate Workflow Extension to add account credentials to users' Wallets. To perform tasks such as create an IMS user, delete an IMS user, and search for IMS users, separate assembly lines will be configured on the IBM Tivoli Directory Integrator (IDI). The assembly lines will be activated by an event handler configured for an IMS naming context.

Once the workflow extension has been added to ITIM, and the assembly lines and event handler configured on IDI, all application accounts provisioned through IBM Tivoli Identity Manager will be single sign-on enabled.

When an administrator creates a user in ITIM along with the ITAM account, a corresponding account is also provisioned in IAM. The user's ITAM logon credentials are automatically populated in the Encentuate Wallet.

The user would just need to log on to the Wallet by presenting one or more authentication factors to AccessAgent, which will then perform automatic sign-on to enterprise applications, including ITAM. Note that the authentication factors can include hardware authentication factors, such as USB token or RFID card.

## Communications between ITIM, ITAM, IMS Server, and AccessAgent




Communication process between ITIM, ITAM, IMS Server, and AccessAgent

❶ When ITIM provisions a new user, it raises an event via DSML to the IBM Tivoli Directory Integrator (IDI). The user-addition assembly line in IDI communicates with the IMS Bridge to create the new IMS user.

❷ The Encentuate Workflow Extension is inserted into the workflow of each application's creation workflow, including ITAM. When ITIM provisions a new ITAM account for the user, the Encentuate Workflow Extension is invoked and passes the ITAM account data to the IMS Server using SOAP over HTTPS, which in turn populates the user's Wallet with the new ITAM account data.

❸ The user logs on to the Wallet by presenting one or more authentication factors to AccessAgent, which will then obtain the Wallet, containing the new ITAM account data, from the IMS Server.

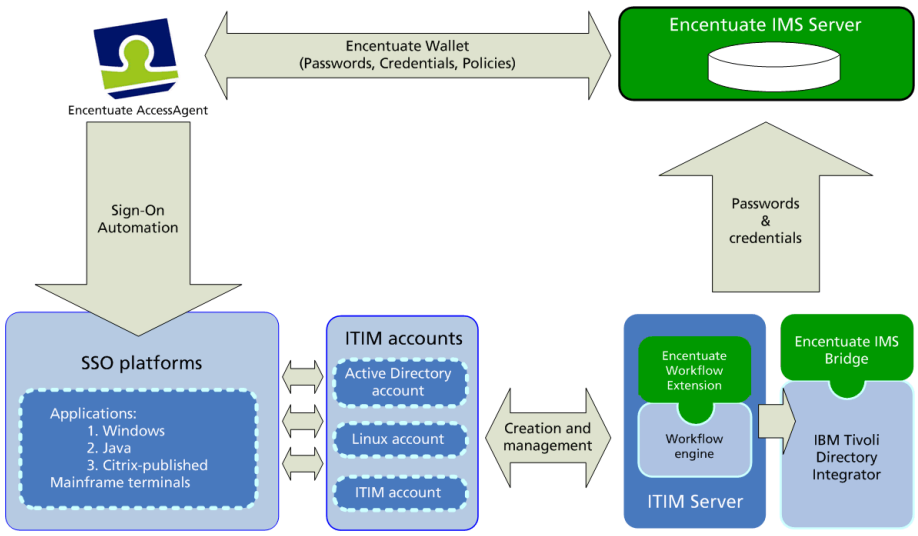
❹ AccessAgent performs sign-on automation for all types of applications: enterprise, personal, certificate-enabled, and any Windows user accounts. AccessAgent auto-fills appropriate user credentials when an application is launched and logs the user onto the application.

When an application integrated with ITAM is launched, AccessAgent auto-fills the ITAM user name and password in the ITAM basic authentication logon prompt. The user does not need to know the ITAM user name and password.

❺ When ITIM de-provisions a user, it raises an event via DSML to IDI. The user-deletion assembly line in IDI communicates with the IMS Bridge to delete the user.

❻ The deleted user can no longer log on to AccessAgent to perform sign-on automation.

## Communications between the ITIM and the IMS Server




Communication process between ITIM and IMS Bridge

The following are the possible communication processes between the IMS Bridge and the IMS Server using Simple Object Access Protocol (SOAP) over HTTPS:

❶ When ITIM provisions new users, it raises an event via DSML to the IBM Tivoli Directory Integrator (IDI). The user-addition assembly line in IDI communicates with the IMS Bridge using Javascript to create IMS users.

❷ The Encentuate Workflow Extension is inserted into the workflow of each application's creation workflow. When ITIM provisions new accounts for an application for users, the Encentuate Workflow Extension is invoked and passes the users' account data to the IMS Server using SOAP over HTTPS, which in turn populates the users' Wallets with the new account data.

❸ The Encentuate Workflow Extension is inserted into the workflow of each application's password-reset workflow. In the event of a password reset for users, the Encentuate Workflow Extension is invoked and passes the new password information to the IMS Server which updates the credentials in the users' Wallets.

❹ When ITIM de-provisions users, it raises an event via DSML to IDI. The user-deletion assembly line in IDI communicates with the IMS Bridge using Javascript to delete the users.

❺ The Encentuate Workflow Extension is inserted into the workflow of each application's deletion workflow. When ITIM de-provisions an enterprise application account, the Encentuate Workflow Extension is invoked and sends the command to the IMS Server, which in turn deletes the application record from the users' Wallets.

❻ When users log on to Encentuate's client software, AccessAgent, the software downloads the users' Wallets from the IMS Server and subsequently performs sign-on automation for all types of applications: enterprise, personal, certificate-enabled, and any Windows user accounts. The AccessAgent will:

- Auto-fill users' credentials into the appropriate application.
- Log the users into the application.

AccessAgent also detects any password change and synchronizes the Wallets on the users' personal computers with the Wallets stored in the IMS Server.

# Minimum requirements

## ITIM

- IBM Tivoli Identity Manager: 4.5.1 and 4.6
- Encentuate IMS Server: 3.0.0.0 and above
- Encentuate AccessAgent: 3.0.3.4 and above

## ITAM

- IBM Tivoli Access Manager: 5.1 and above
- Encentuate IMS Server: 3.0.0.0 and above
- Encentuate AccessAgent: 3.0.3.4 and above
- Encentuate AccessStudio: 3.0.1.2 and above

# Using the integration package

The ITIM integration package contains the following folders:

- **provisioningbridge [Encentuate provisioning bridge]**
  - bin
  - config [configuration files]
  - docs [documentation and notes]
  - lib [Java libraries]
    - commons-launcher [Java libraries for commons-launcher]
    - license [license text files]
- **wfe [Encentuate workflow extensions]**
  - config [configuration files]
    - ImsService [ITIM configuration files for ImsService account]
  - **docs [documentation and notes]**
  - **lib [Java libraries]**

Use the integration package for the correct ITIM version.

To begin, put these folders and files (preserving the folder structure) into a desired folder on the ITIM server (e.g., "C:\Encentuate").

Be sure to complete the steps in Configuring The IMS Server.

# Configuring the IBM Tivoli Directory Integrator (IDI)

IDI configuration is an automated process, facilitated by placing files required by the IMS Bridge in the corresponding folder(s) and restarting the IDI.

*To configure the IBM Tivoli Directory Integrator:*

1. Place all associated JAR files from **C:\Encentuate\provisioningbridge\lib** into the **<IDI_INSTALL_DIR>\jars\provisionagent** directory, except for **bcprov.jar**.
2. Place **bcprov.jar** in **<IDI_INSTALL_DIR>\jvm\jre\lib\ext** folder.

❸ Create a keystore that contains the IMS Server's SSL certificates as trusted certificate entries. You can use Internet Explorer to download the IMS Server's SSL certificate into the Windows certificate store by visiting https://imsserver/ where **imsserver** is the IMS Server's hostname, and then click the **SSL lock** icon to view certificate.

Click the **Details** tab and proceed to **Copy to File** using the Base-64 encoded X.509 (.CER) format. Use the **keytool.exe** bundled with IBM JDK, which is distributed with WebSphere Application Server, to import the IMS Server certificate. **keytool.exe** can be found in the **<JAVA_HOME>\jre\bin** directory.

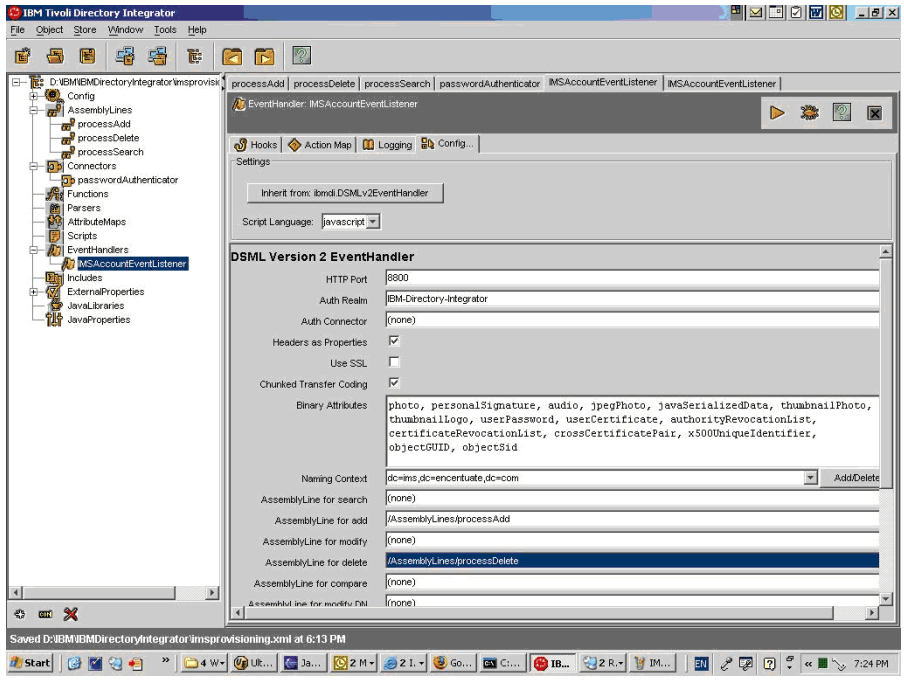
For more information on the keytool utility, see Keytool.

❹ Edit **<IDI_INSTALL_DIR>/global.properties** to specify truststore and keystore information. In the current release, only jks-type is supported.

```
# Keystore file information for the server authentication.

# It is used to verify the server's public key.

# example

javax.net.ssl.trustStore=C:\Encentuate\provisioningbridge\confi
g\truststore.jks

javax.net.ssl.trustStorePassword=password

javax.net.ssl.trustStoreType=jks



# Keystore file information for the client authentication.

# It is used to provide the public key of the IBM Tivoli
Directory Integrator

  to the server if the server requests the client
authentication.

# example

javax.net.ssl.keyStore=C:\Encentuate\provisioningbridge\config\
truststore.jks

javax.net.ssl.keyStorePassword=password

javax.net.ssl.keyStoreType=jks
```

If these keys are not configured yet, you can set both truststore and keystore to the same one that contains the IMS Server certificate. Otherwise, you need to import the IMS Server certificate to the truststore specified in **javax.net.ssl.trustStore**.

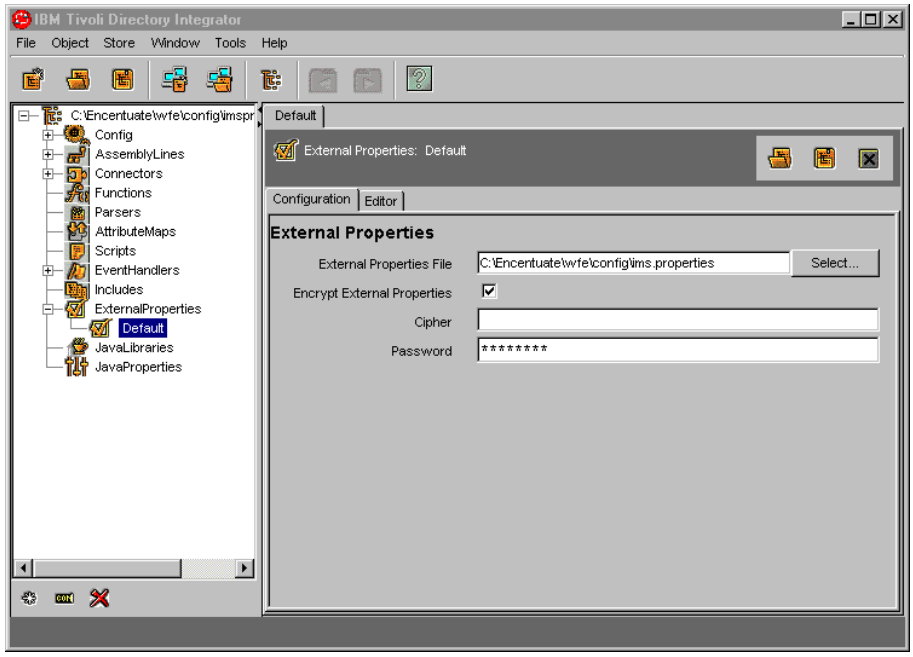
5. Launch IDI and open the IMS Bridge configuration file located in **C:\Encentuate\wfe\config\imsprovisioning.xml**. If there are no plans to protect the communications between ITIM and IDI with SSL, clear the **Use SSL** checkbox. If SSL will be used, the SSL certificate of the ITIM server must be generated and trusted by IDI, and vice-versa.




IDI Configuration Editor

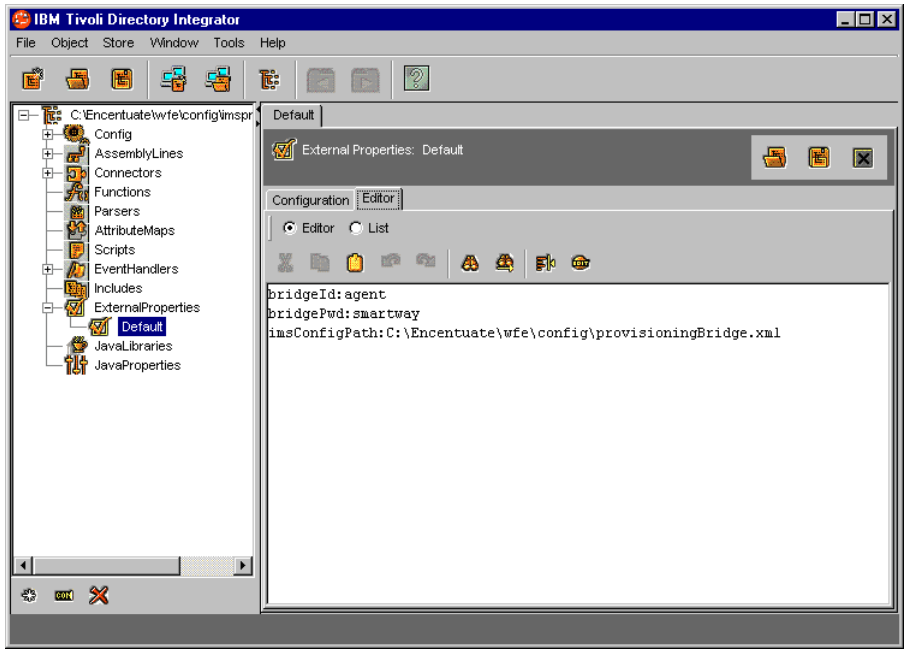
6. Click *ExternalProperties >> Default* in the left panel. The provisioning bridge running in IDI needs to authenticate with IMS Server to demonstrate its authenticity. The authentication mechanism used is password-based logon on an SSL channel. The logon credential should be specified in an encrypted Java properties file created via this IDI ExternalProperties function.

   Create a new **ims.properties** file. Mark the **Encrypted External Properties** checkbox and choose a password for the file. You can specify the cipher, depending on its availability in IDI. Refer to the IDI documentation on the available ciphers.




IDI ExternalProperties

❼ Click the **Editor** tab to edit the external properties file.




IDI ExternalProperties Editor

❽ Click the **Play** button (see IDI Configuration Editor) to run **IDI Server** with the IMS Bridge configuration.

# Configuring the IBM Tivoli Identity Manager

To configure ITIM, you need to define and add a data model to the ITIM data store, then configure ITIM for management of the newly-defined account type.

The following sections describe the step-by-step ITIM configuration process.

## Setting up the initial environment

*To set up the initial environment:*

1. Copy **encwfe.properties** from the **C:\Encentuate\wfe\config directory to <ITIM_INSTALL_DIR>\data** directory. The configurations must be correct for a particular deployment.

2. Open the **Properties.properties** file in the same directory and add one entry to point to **encwfe.properties**.

   ```
   # Provisioning bridge credential

   enc.workflowextension=encwfe.properties
   ```

## Configuring the provisioning bridge

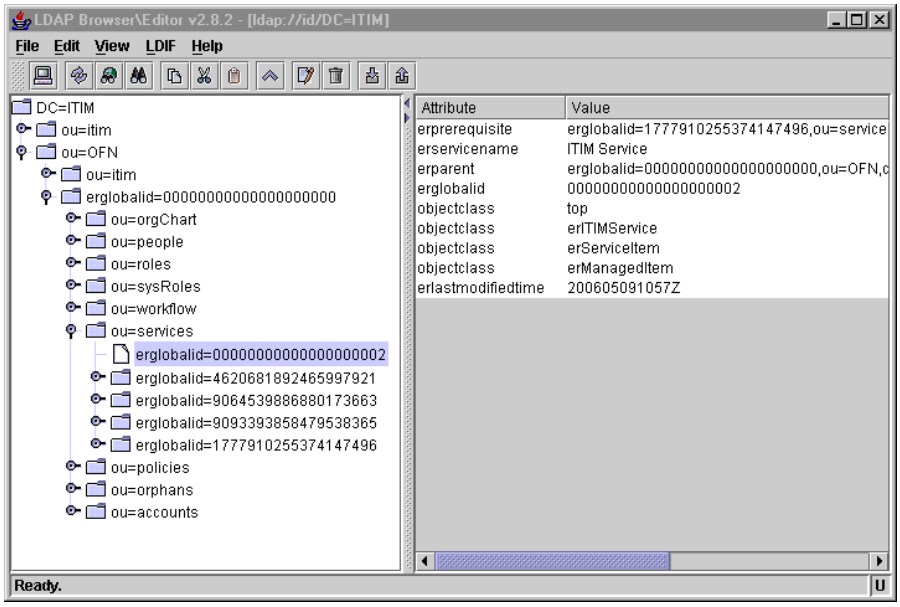
*To configure the provisioning bridge:*

1. Open **provisioningBridge.xml** in **C:\Encentuate\provisioningbridge\config** directory and make the appropriate changes to suit the local environment. Refer to **C:\Encentuate\provisioningbridge\docs\ConfigurationGuide.txt** for details.

2. The **provisioningbridge.authenticationService.mapping** configuration key of **provisioningBridge.xml** maps Distinguished Names of registered ITIM services to IMS authentication services.

   To see the Distinguished Names of all registered ITIM services, use any LDAP-compatible browser to log on to ITIM back-end data store.

   You can find all services in ou=services. In the example shown, the Distinguished Name of the ITIM service is **erglobalid=00000000000000000002,ou=services,erglobalid=00000000000000000000,ou=OFN,dc=ITIM**.




Identifying ITIM services using LDAP browser

❸ Copy **encwfe.jar** and **log4j-1.2.9.jar** from **C:\Encentuate\wfe\lib** directory to the **<WEBSPHERE_HOME>\AppS-erver\installedApps\<SERVER_NAME>\enRole.ear\app_web.war\WEB-INF\lib** directory. If there is no directory, create a new one.

# Defining a data model

The files in the directory **C:\Encentuate\wfe\config\ImsService** define a data representation for an ImsService account on ITIM.

The following files are included:

- **schema.dsml** defines the directory syntax for the account and service classes.
- **resource.def** is the resource definition for the creation of a service profile.
- **CustomLabels.properties** defines labels for the forms displayed in the user interface.
- **imsaccount.xml** represents account entries associated with the service of type imsservice.
- **imsservice.xml** represents a service in ITIM to manage Encentuate IMS accounts.

The **schema.dsml** file contains the definitions of IMS attributes and object classes for the account, service, and a group object in DSML format.

These are described in the following table:

| Entity Type | Object Class | Description |
|---|---|---|
| Service | imsservice | This represents a service in ITIM to manage Encentuate IMS accounts. |
| Account | imsaccounts | This represents account entries associated with the service of type imsservice. |

Entities defined for data model

The attributes associated with the service are described in the following table. The labels are given in CustomLabels.properties.

| Attribute Name | Label | Required | Description |
|---|---|---|---|
| erservicename | Service Name | Yes | The name of the service to display on the ITIM user interface. |
| erurl | URL | Yes | The URL that IDI is listening on. |
| eruid | User ID | Yes | The principal used for authentication of ITIM by IDI. |
| erpassword | Password | Yes | The password used for authentication of ITIM by IDI. |

Definition of IMS service attributes

The attributes associated with the account are described in the following table:

| Attribute Name | Label | Required | Description |
|---|---|---|---|
| eruid | User ID | Yes | The ID used to identify the account users. |
| erpassword | Password | Yes | The password that the managed resource uses for authenticating its users. |

Definition of IMS account attributes

The service and accounts profiles are defined in the resource definition file, **resource.def**. This file also contains an attribute for the factory for handling the protocol, as well as a list of service properties to send with requests.

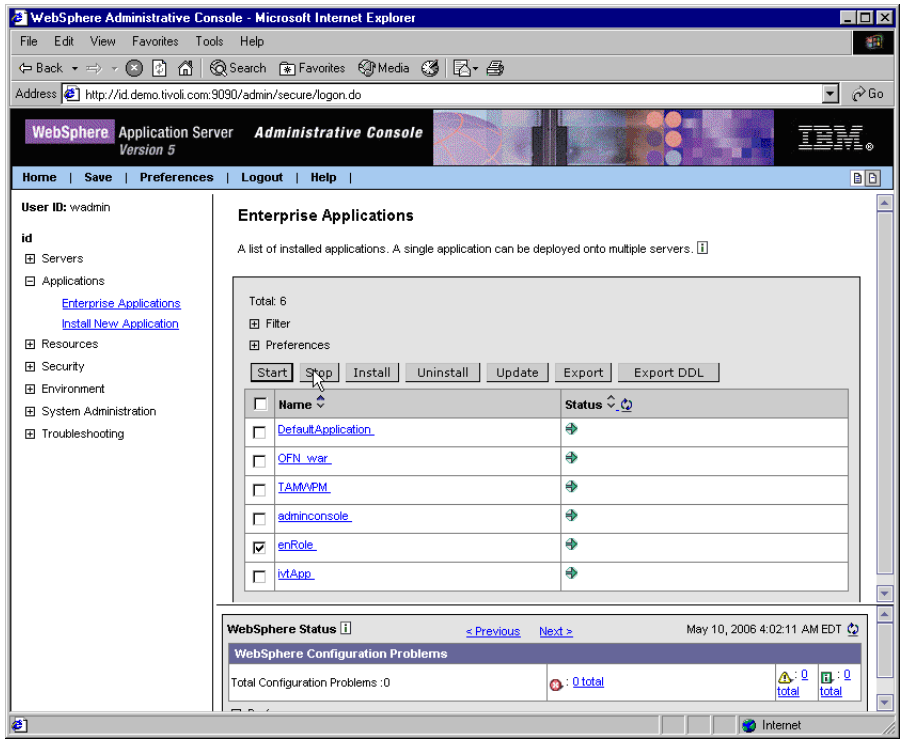
### *To load the data definitions into ITIM:*

❶ Copy the directory **C:\Encentuate\wfe\config\ImsService** and its contents to the location **<ITIM_INSTALL_DIR>\data\remote_resources**.

❷ Bring up a command line interface and change directories to the **<ITIM_INSTALL_DIR>\bin\win**.

3. Execute the command `config_remote_services -profile ImsService`.

4. Restart the ITIM server through the WebSphere Administrative Console. ITIM should be listed as **enRole** in WebSphere.




Restarting ITIM server from WebSphere Administrative Console

Use the directory administration console to verify if the IMS schema has been imported successfully. Any errors will appear in the ITIM log and the directory log (for schema import problems).

*Use the same steps in this procedure for a UNIX platform.*

# Configuring the IMS Service in ITIM

***To configure the IMS Service in ITIM:***

1. Go to *Provisioning >> Service Management*.

2. Add a new service of type ImsService.

❸ Add the following values for the IMS service parameters.

| Parameter | Value | Explanation |
|---|---|---|
| Service Name | ImsService | A value to display on the ITIM user interface. |
| URL | http://localhost:8800 | This is needed to locate IDI. |
| User ID | agent | The principal used for ITIM to authenticate to IDI. |
| Password | smartway | The password used for ITIM to authenticate to IDI. |
| Naming Context | dc=ims | Used to relate requests to the correct context within IDI. |
| Category | Account | Type of entity for use with ITIM data services APIs. This is the appropriate value for account management. |

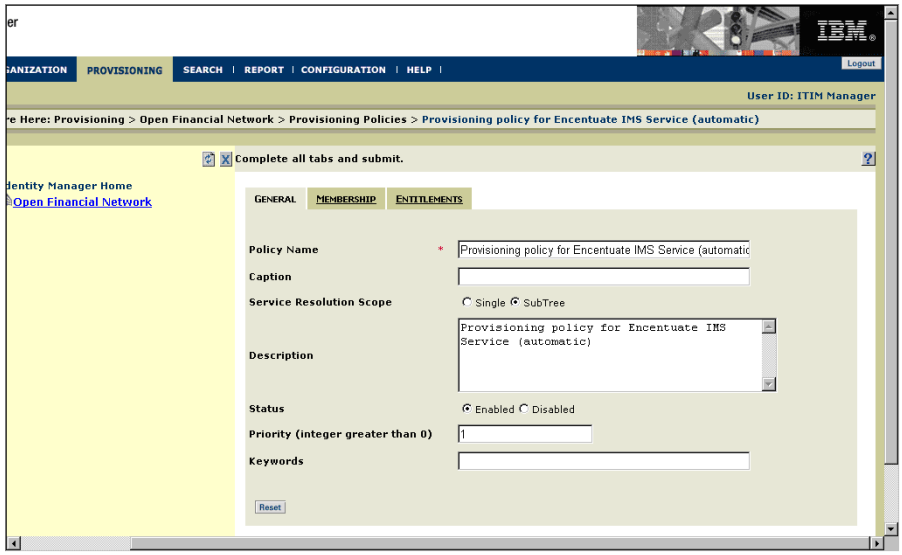
IMS service parameters

Refer to the sample screenshot of the service form.


Custom service instance detail form

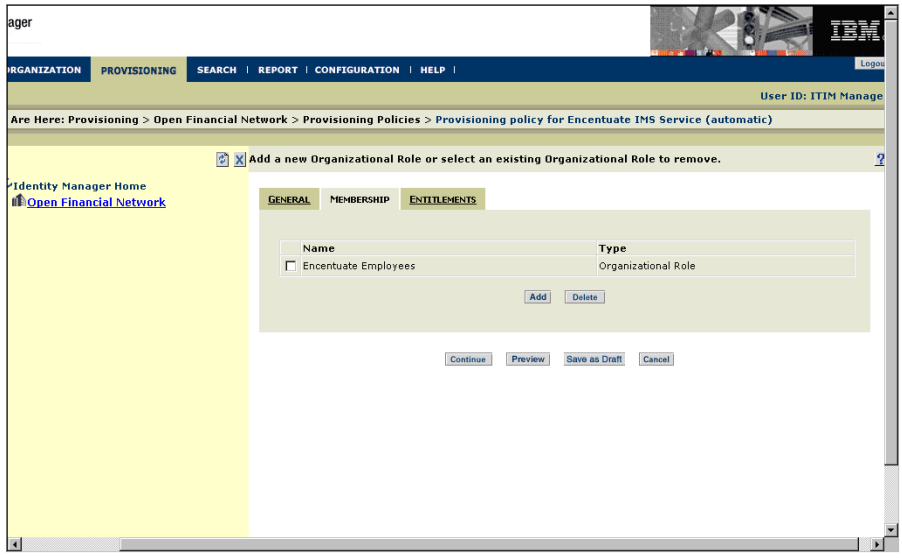
Make sure the connection test passes before proceeding with the remaining portions of this guide.

❹ Create a new provisioning policy for the new service by going to *Provisioning >> Define Provisioning Policies*.



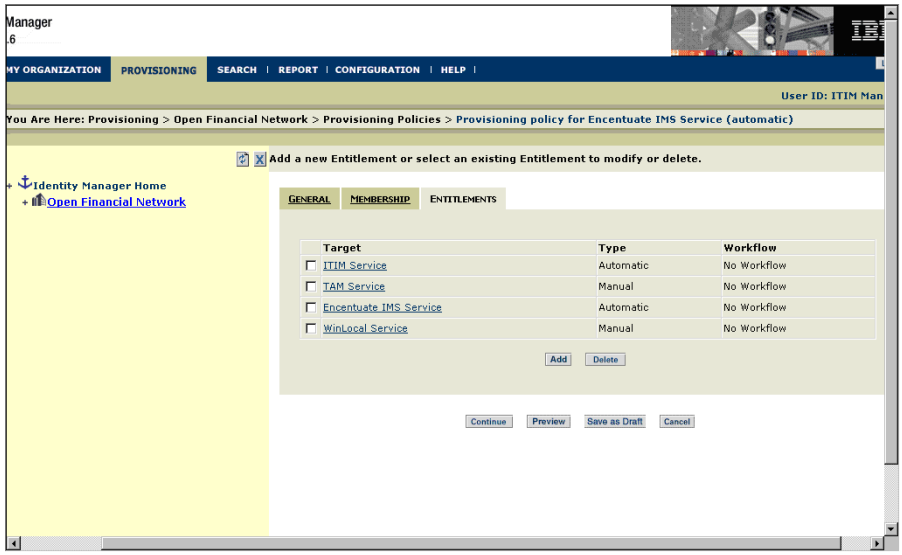

Create a provisioning policy

❺ Specify the Membership for this policy.




Defining membership for a provisioning policy

❻ Specify the Entitlements for this policy. You can set the accounts to be created automatically once an IMS service account is created.




Defining entitlements for a provisioning policy

# Configuring Encentuate workflow extensions

This section describes the steps to adding custom workflow extensions to be used as workflow objects within the IBM Tivoli Identity Manager.

## Adding a workflow extension

Edit the workflowextensions.xml file under <ITIM_INSTALL_DIR>\data directory to add a workflow extension. Add the following workflow extension:

```
<ACTIVITY ACTIVITYID="encAddAccount" LIMIT="600000">

  <IMPLEMENTATION_TYPE>

    <APPLICATION
CLASS_NAME="encentuate.bridges.wfe.EncentuateCltAppExtension"
METHOD_NAME="encAddAccount"/>

  </IMPLEMENTATION_TYPE>

  <PARAMETERS>

    <IN_PARAMETERS PARAM_ID="owner" RELEVANT_DATA_ID="owner"
TYPE="Person"/>

    <IN_PARAMETERS PARAM_ID="service"
RELEVANT_DATA_ID="service" TYPE="Service"/>

    <IN_PARAMETERS PARAM_ID="account"
RELEVANT_DATA_ID="account" TYPE="Account"/>
```

```
  </PARAMETERS>

</ACTIVITY>

<ACTIVITY ACTIVITYID="encChangePassword" LIMIT="600000">

  <IMPLEMENTATION_TYPE>

    <APPLICATION
CLASS_NAME="encentuate.bridges.wfe.EncentuateCltAppExtension"
METHOD_NAME="encChangePassword"/>

  </IMPLEMENTATION_TYPE>

  <PARAMETERS>

    <IN_PARAMETERS PARAM_ID="account" RELEVANT_DATA_ID="Entity"
TYPE="Account"/>

  </PARAMETERS>

</ACTIVITY>

<ACTIVITY ACTIVITYID="encDeleteAccount" LIMIT="600000">

  <IMPLEMENTATION_TYPE>

    <APPLICATION
CLASS_NAME="encentuate.bridges.wfe.EncentuateCltAppExtension"
METHOD_NAME="encDeleteAccount"/>

  </IMPLEMENTATION_TYPE>

  <PARAMETERS>

    <IN_PARAMETERS PARAM_ID="account" RELEVANT_DATA_ID="Entity"
TYPE="Account"/>

  </PARAMETERS>

</ACTIVITY>

<ACTIVITY ACTIVITYID="hasImsAccount" LIMIT="600000">

  <IMPLEMENTATION_TYPE>

    <APPLICATION
CLASS_NAME="encentuate.bridges.wfe.EncentuateCltAppExtension"
METHOD_NAME="hasImsAccount"/>

  </IMPLEMENTATION_TYPE>

  <PARAMETERS>

    <IN_PARAMETERS PARAM_ID="owner" RELEVANT_DATA_ID="owner"
TYPE="Person"/>
```

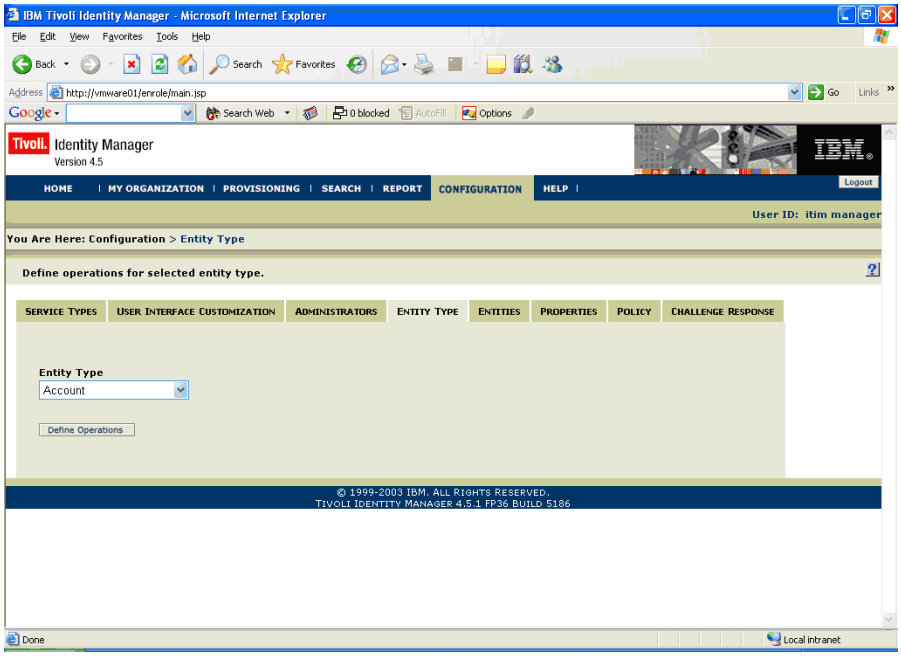
```
    <OUT_PARAMETERS PARAM_ID="skipFlag"
RELEVANT_DATA_ID="skipFlag" TYPE="String"/>

  </PARAMETERS>

</ACTIVITY>
```

# Defining workflows with extensions

To define workflow extensions, make sure that ITIM is running.

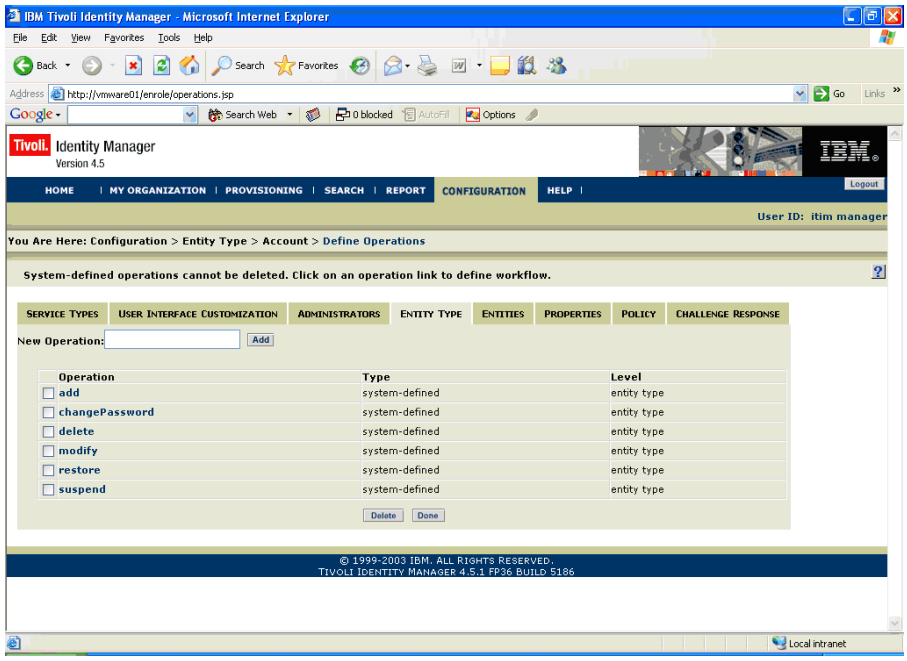
*To define workflow extensions:*

1. Log on to ITIM.
2. Select *Configuration >> Entity Type*.



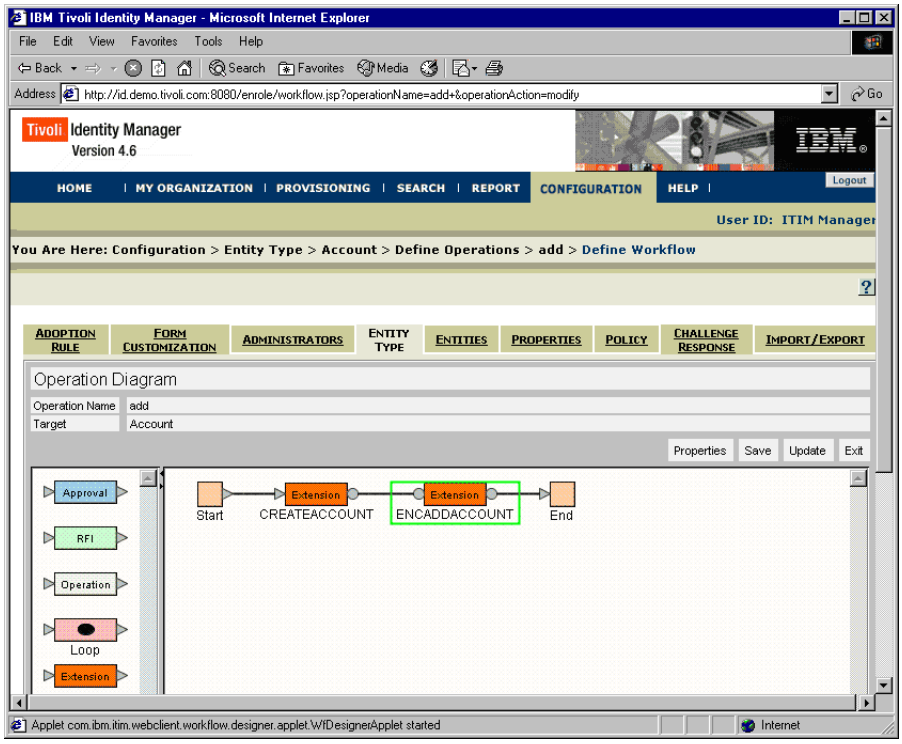

Configure entity type

3. Select **Account** as **Entity Type** and click **Define Operations**.




Define operations for Account

❹ Click **Add**. The operation diagram is displayed. Provide the same changes shown in the following screenshot.

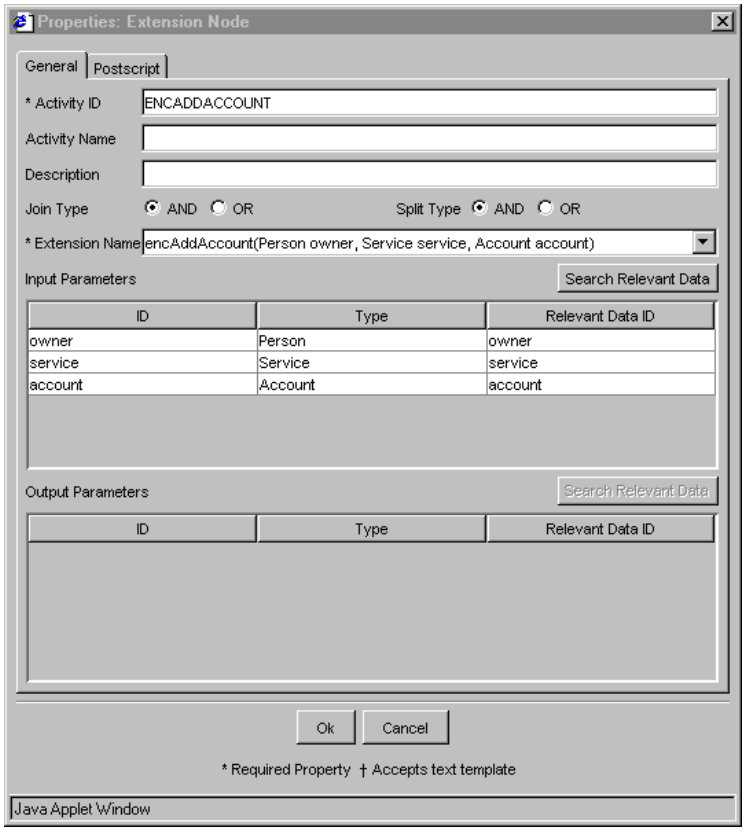



Define add account operation

❺ Remove the transition from CREATEACCOUNT to End.

❻ Add a new extension node between CREATEACCOUNT to End.

❼ Double-click on the new Extension node. A pop-up window displays all the extensions registered using **workflowextensions.xml**.




Define ENCADDACCOUNT

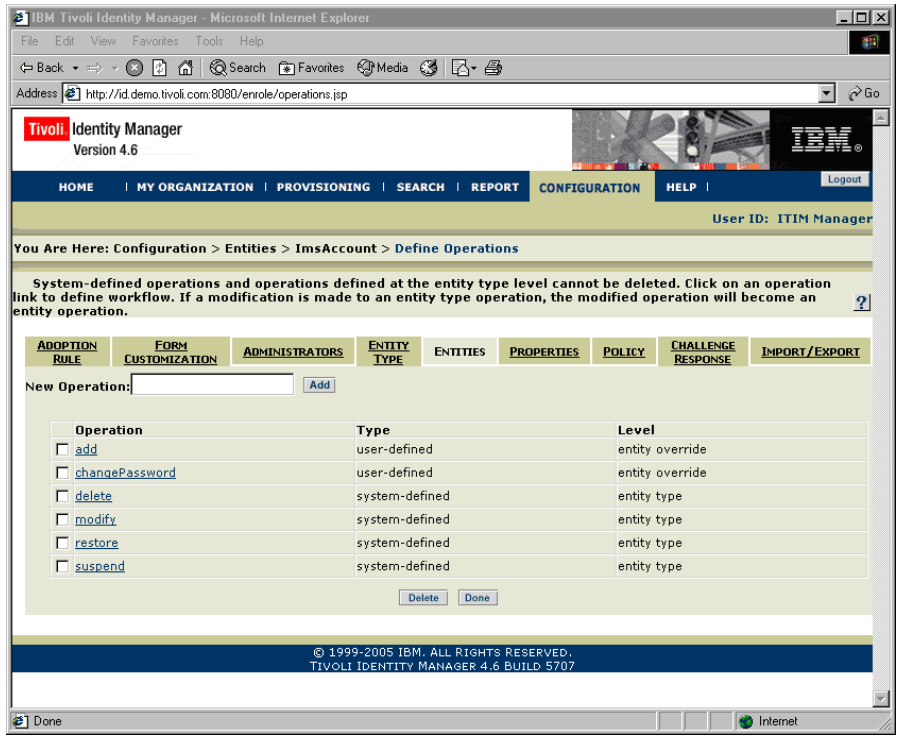
❽ Select the **Extension Name** as encAddAccount and fill in the **Activity ID** with ENCADDACCOUNT.

❾ Click **Ok** and attach the transitions to the newly-added extension.

❿ Click **Save**.

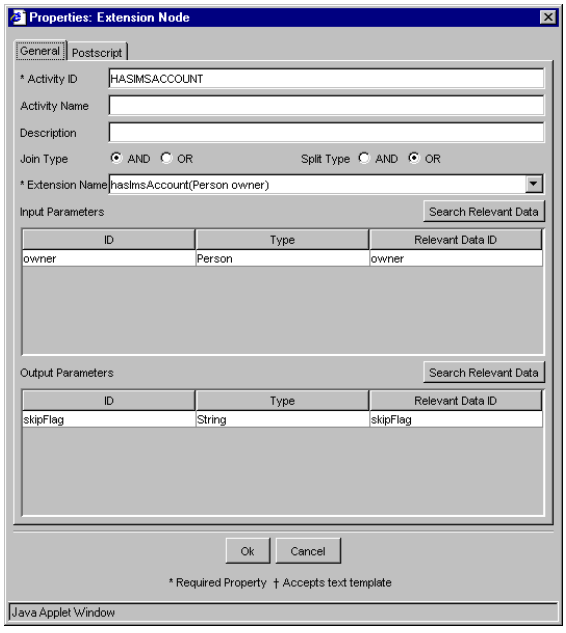
⓫ Repeat the above for changePassword and delete operations.

⓬ Define ImsAccount operations by going to *Configuration >> Entities >> ImsAccount*.



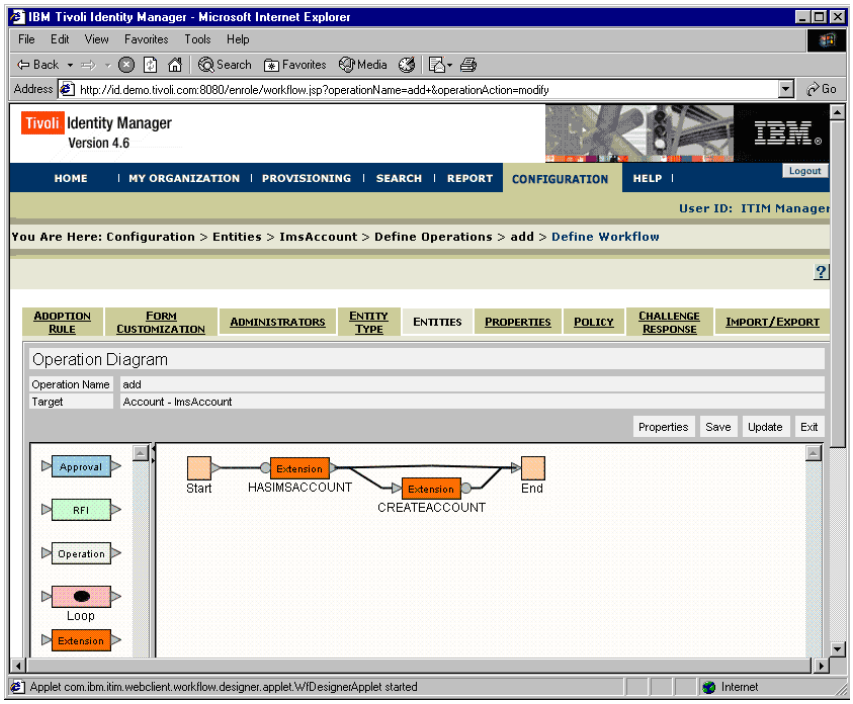

Define ImsAccount operations

13. To prevent the creation of more than one ImsAccount for a Person in ITIM, open add workflow of ImsAccount and insert the extension hasImsAccount to ensure only one IMS account is created for every Person in ITIM. Use **OR** for Split Type.



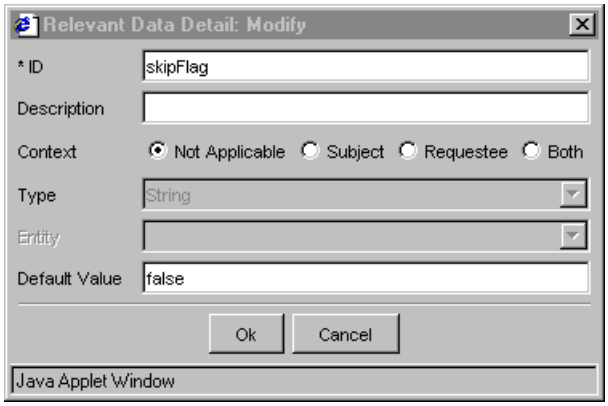

Define HASIMSACCOUNT extension

⓮ The final workflow will look something like this, after createAccount is added too:

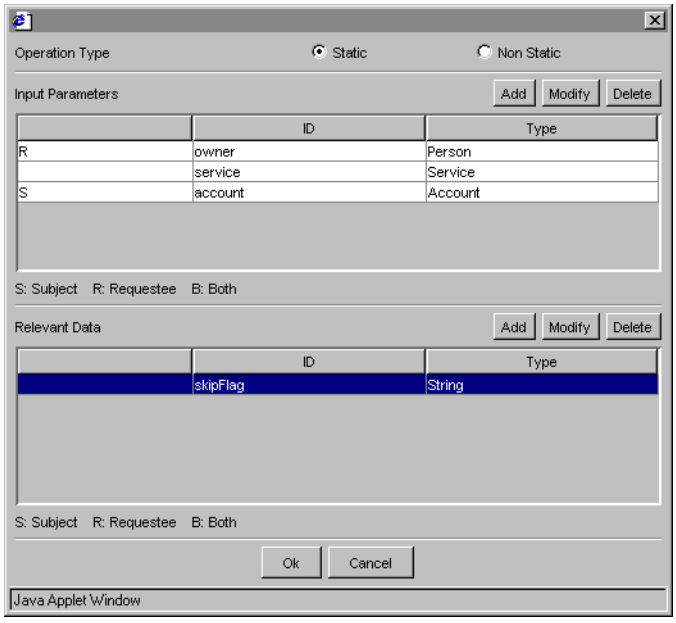



Workflow for add ImsAccount

⓯ Click **Properties** and **Add** to create a new skipFlag property to determine the workflow path:

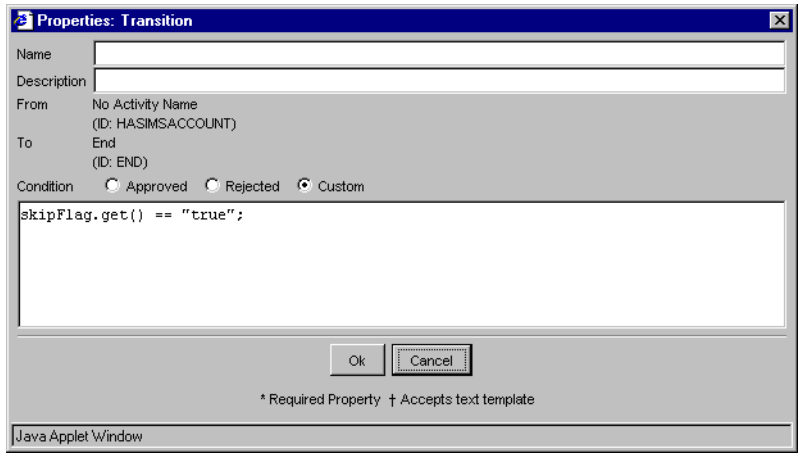



Define skipFlag



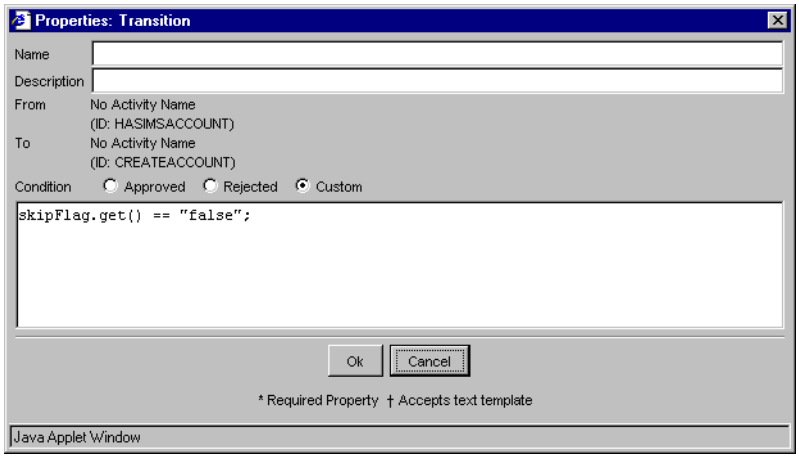

Properties list after skipFlag is added

16. Double-click on the transition connecting HASIMSACCOUNT to End and key in the JavaScript condition skipFlag.get() == "true";




Checking for skipFlag == true

17. Double-click on the transition connecting HASIMSACCOUNT to CREATEACCOUNT and key in the JavaScript condition skipFlag.get() == "false";



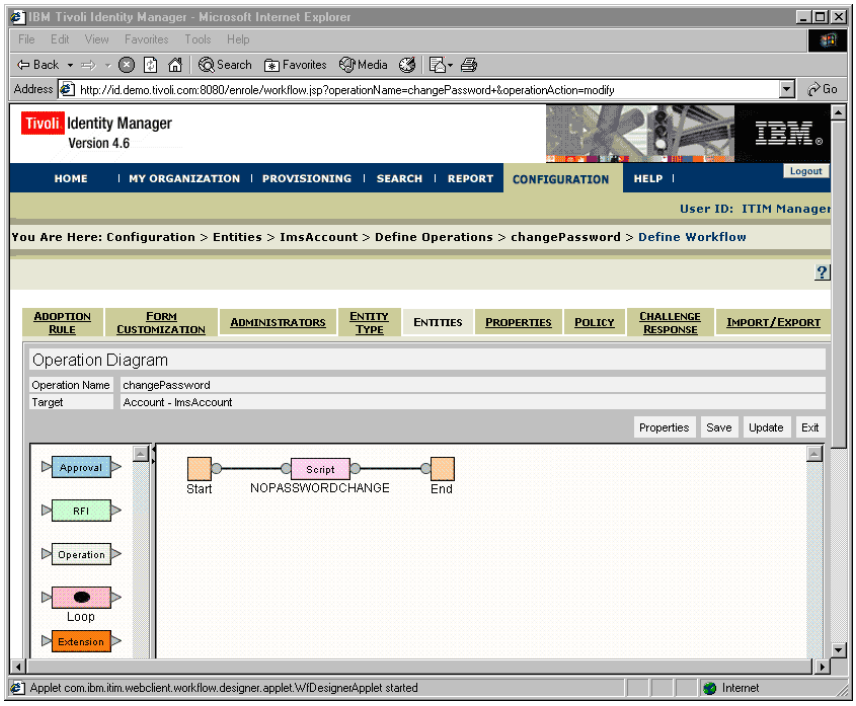

Checking for skipFlag == false

⓲ Ensure that the **Join Type** is **OR** for the End node.

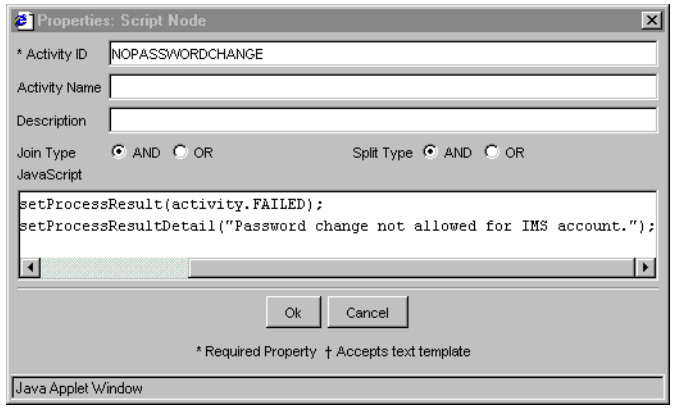

Properties for End node

⓳ To disable password change for ImsAccount, open changePassword workflow of ImsAccount and remove the default extension for password change. Add a new extension NOPASSWORDCHANGE with the following JavaScript:

```
WorkflowRuntimeContext.setProcessResult(activity.FAILED);

WorkflowRuntimeContext.setProcessResultDetail("Password change
not allowed for IMS account.");
```




Define passwordChange workflow




JavaScript for NOPASSWORDCHANGE

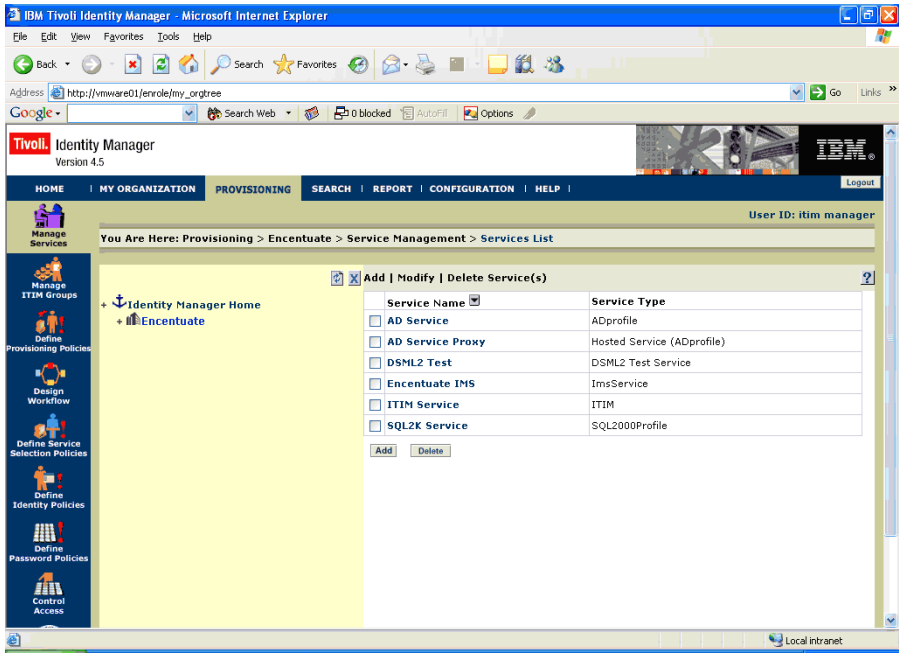
20. Click **Save**.

## Setting service prerequisites

As a prerequisite, all application services in ITIM can only be provisioned after user accounts are provisioned in IMS Server. Otherwise, sign-on automation will not work.

*To set a service prerequisite:*

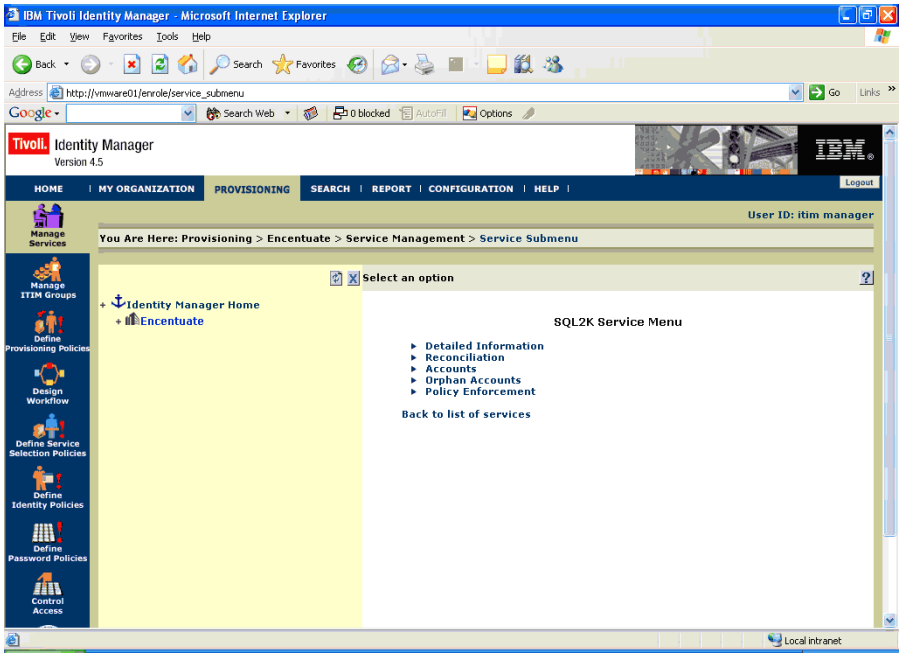
❶ In ITIM, select *Provisioning >> [Organization Name] >> Service Management >> Services List*.




Setting service prerequisite by selecting each service

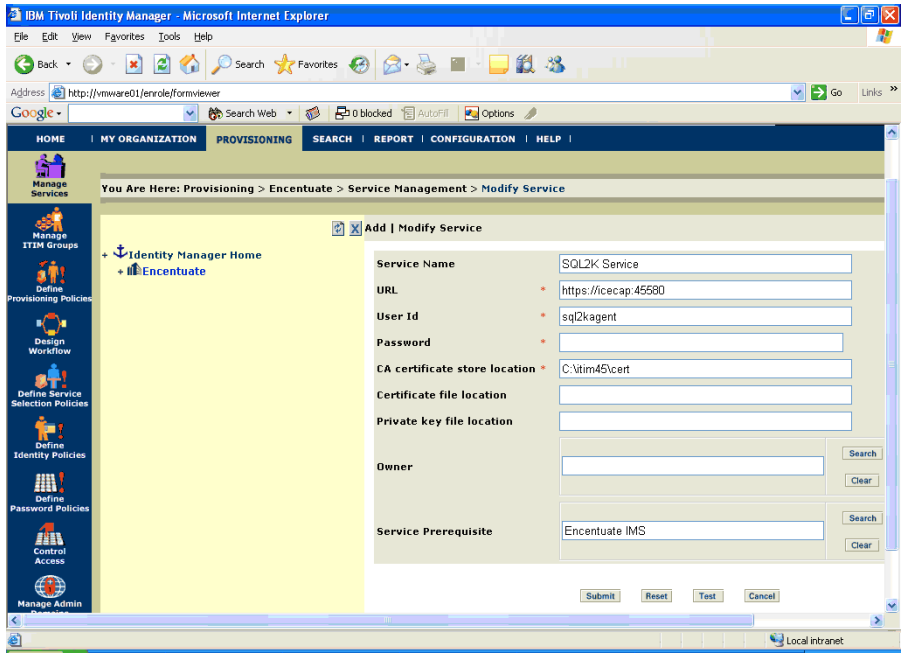
❷ Mark the checkbox of the service. For example: SQL2K Service.

❸ Click **Modify**.




Setting service prerequisite for a particular service

❹ In the **Service Submenu**, click **Detailed Information**.

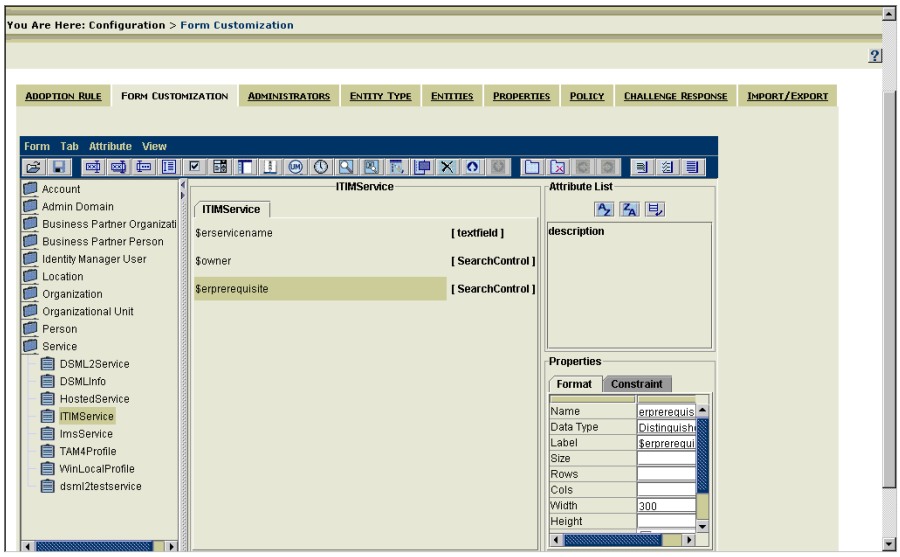



Set IMS Service as prerequisite

❺ At the bottom of the **Modify Service** panel, click **Search next** to the Service Prerequisite field and select: Encentuate IMS.

❻ Click **Submit**. Repeat with other services as needed.

❼ To create a Service Prerequisite field for ITIM Service, go to *Configuration >> Form Customization* and add a new $erprerequisite field for the ITIMService.




# Provisioning setup and maintenance

*To provision a new user (ITIM):*

❶ ITIM provides the IMS user names and initial passwords to IDI, which in turn will provision the IMS users through the IMS Bridge. The passwords can be randomly-generated, automatically-assigned to users' Employee IDs, or manually assigned through ITIM.

❷ IDI connects to the IMS Bridge to provision the IMS users using the IMS user name and password generated by ITIM. The users' Wallets will also be initialized on the IMS Server at this time such that account credentials can then be added to the Wallets.

*IMS user accounts should be created before other application accounts. Account credentials cannot be added to the user's Wallet before the users are provisioned on the IMS Server.*

❸ The users log on with their IMS user names and initial passwords when they use AccessAgent for the first time. They will be prompted to change the initial password and the Wallet containing the provisioned account credentials will be downloaded from the IMS Server.

*To add an application account (ITIM):*

❶ The ITIM Workflow Engine invokes the Encentuate Workflow Extension with the user names and the application credentials.

❷ The Encentuate Workflow Extension connects to the IMS Server to add the application and credential information to the users' Wallets.

❸ The next time users access AccessAgent, the AccessAgent will have the necessary credentials in the Wallet to automate sign-on to the new application. Applications can therefore be added without having to inform users of the new credentials. Users just need to sign-on to AccessAgent.

*To reset an application password (ITIM):*

❶ The ITIM Workflow Engine invokes the Encentuate Workflow Extension with the user names and the new application passwords.

❷ The Encentuate Workflow Extension connects to the IMS Server to update the application passwords in the users' Wallets.

❸ The next time users log on to AccessAgent, the AccessAgent will have the updated application passwords in the Wallets to automate sign-on to the applications. Administrators can reset application passwords directly without notifying each user. Users just need to sign-on to Encentuate to log on to the application.

*To delete an application account (ITIM):*

❶ The ITIM Workflow Engine invokes the Encentuate Workflow Extension with the application accounts to delete.

❷ The Encentuate Workflow Extension connects to the IMS Server to delete the application accounts from the users' Wallets.

❸ The next time users access AccessAgent, AccessAgent will no longer have access to the deleted application's credentials in the Wallets, and cannot sign-on to the application on the users' behalf. Applications can be removed centrally and all access can be terminated automatically.

*To de-provision users (ITIM):*

❶ ITIM provides the IMS user ID of the users to be de-provisioned to IDI which will de-provision the users.

❷ IDI connects to the IMS Bridge and revokes the IMS users. The revocation of the IMS users will invalidate both the users' accounts and the users' Wallets on the server.

❸ If the users attempt to log on using AccessAgent, the log on will fail and Wallets that have been cached locally by AccessAgent will be revoked.

# Configuring ITAM

Use the IBM Tivoli Access Manager to manage your AccessProfiles. This is an optional configuration for organizations that use ITIM with Encentuate IAM for provisioning.

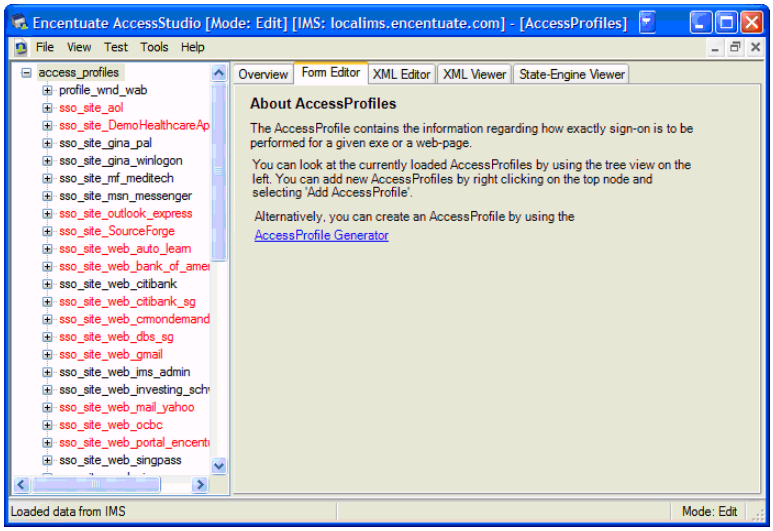
## Creating AccessProfiles for ITAM

Refer to the configuration steps in this section to understand how to create an AccessProfile for ITAM. With this configuration, AccessAgent can recognize the ITAM basic authentication logon prompt, and auto-fill it with the user's ITAM user name and password.

Before using AccessStudio, log on to AccessAgent as Administrator of IAM.

*To create AccessProfiles for ITAM:*

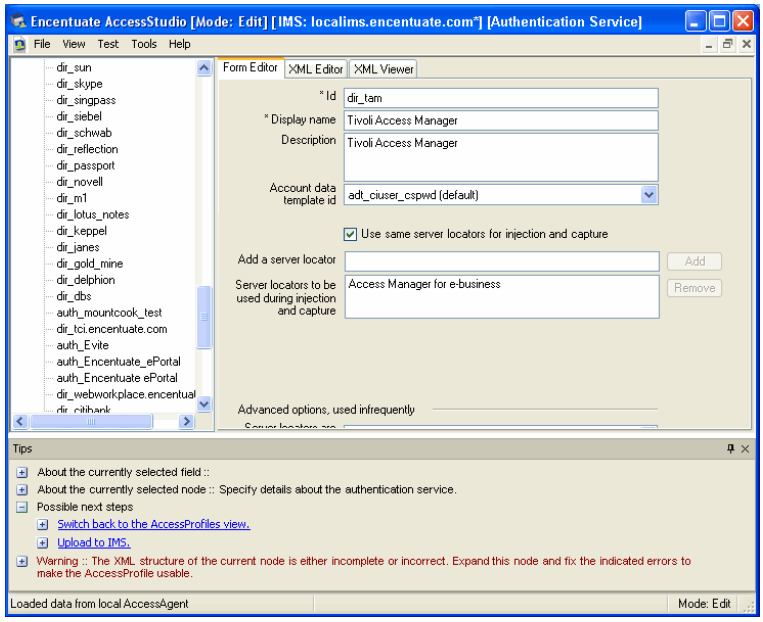
❶ Launch AccessStudio. Click on *File >> Import data from IMS* to download the latest AccessProfiles from IMS Server.




AccessStudio with data imported from IMS Server

❷ Click on *View >> Authentication Services* to go to the authentication services view. In the left panel, right-click the authentication_services node and select **Add Authentication Service**. A new authentication service will be displayed in the right panel.

❸ Enter **dir_tam** in the **Id** field, **Tivoli Access Manager** in the **Display Name** field, and **Tivoli Access Manager** in the **Description** field. Enter **Access Manager for e-business** to the **Add server locator** field. Click **Add** button and the "Access Manager for e-business" appears on the **Server locators to be used during injection and capture** field.

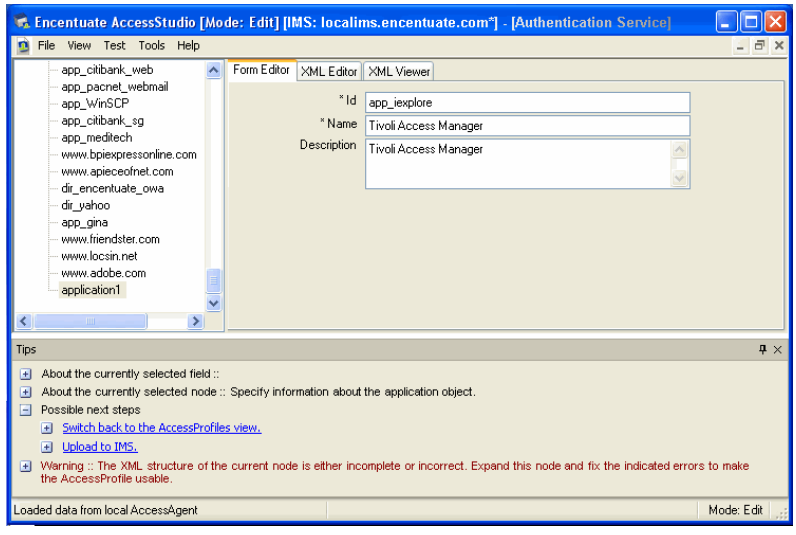



Create authentication service

❹ In the left panel, right-click the node representing the new authentication service and select **Upload to IMS**. This will upload the newly-created authentication service (dir_tam) to the IMS Server.

❺ Click on *View >> Applications* to go to the applications view. In the left panel, right-click the applications node and select **Add Application**. A new application will be displayed in the right panel.

❻ Enter **app_iexplore** in the **Id** field, **Tivoli Access Manager** in the **Display Name field**, and **Tivoli Access Manager** in the **Description** field.




Create application

*The term "iexplore" is used because the application that shows the ITAM basic authentication logon prompt is actually Internet Explorer.*

7. In the left panel, right-click the node representing the new application and select **Upload to IMS**. This will upload the newly-created application (app_iexplore) to the IMS Server.

8. Click on *View >> AccessProfiles* to go to the AccessProfiles view. In the left panel, right-click the access_profiles node and select **Add AccessProfile**. A new AccessProfile will be displayed in the right panel.

   An AccessProfile can now be created for the ITAM basic authentication logon prompt, which is displayed by the Internet Explorer application. It should make use of the ITAM authentication service (dir_tam) and ITAM application (app_iexplore) created earlier.

   Refer to the Encentuate AccessStudio Guide for details.

9. Alternatively, the sample AccessProfile for ITAM can be used. To upload this sample AccessProfile, click on the **XML Editor** tab to switch to the XML Editor view so that the sample AccessProfile can be pasted.

   Open the sample AccessProfile file (**sso_site_wnd_iexplore.xml**) in a text editor (e.g., Notepad) and copy all its contents. In the XML Editor, right-click any text in the editor and select **Select All**. Then, right-click any text in the editor again and select **Paste**.

10. In the left panel, right-click the node representing the new AccessProfile and select **Upload to IMS**. This will upload the newly-created authentication service (**sso_site_wnd_iexplore**) to the IMS Server.

# Configuring ITAM as an enterprise authentication service

Refer to the configuration steps in this section to understand how to configure ITAM as an enterprise authentication service in the IMS Server. With this configuration, AccessAgent can manage ITAM as an enterprise authentication service. Audit logs are submitted to the IMS Server when users log on to ITAM. The AccessAdmin Web interface is used for configuring the IMS Server.

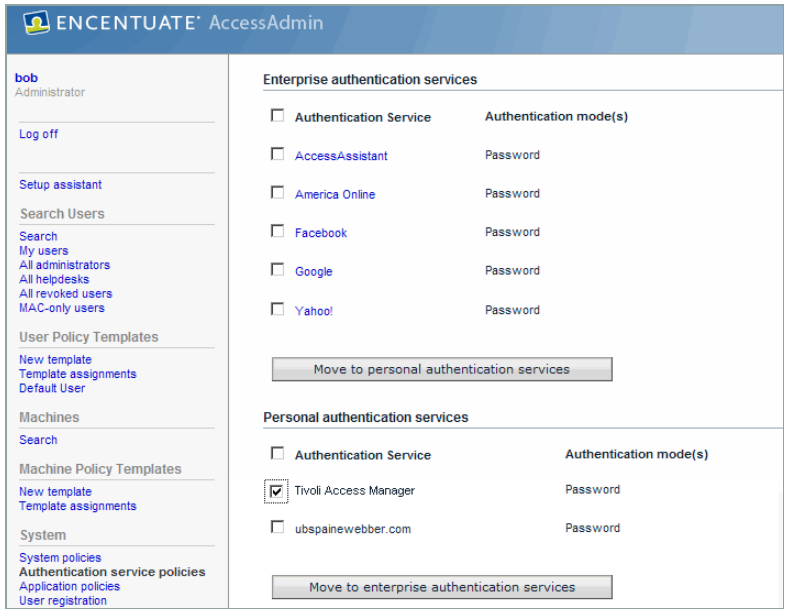
Before using AccessAdmin, log on to AccessAgent as Administrator of IAM.

### *To configure ITAM as an enterprise authentication service:*

1. Launch AccessAdmin (usually https://imsserver where **imsserver** is the IMS Server's hostname).

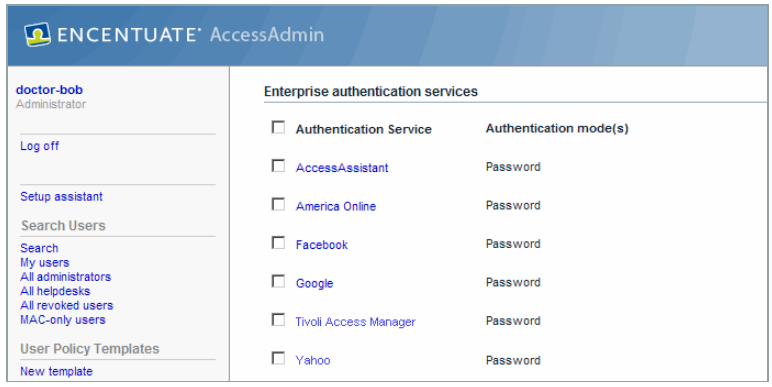
❷ Click on **Authentication service policies** in the left panel. The current list of authentication services will be shown in the right panel.

❸ In the right panel, under **Personal Authentication Services**, look for **Tivoli Access Manager**. Mark the checkbox and click the **Move to enterprise authentication services** button.




Click Move to enterprise authentication services

Tivoli Access Manager should now be moved to the list of enterprise authentication services.




ITAM as enterprise authentication service

# M-Tech ID-Synch

Encentuate IAM and M-Tech ID-Synch integration provides an immediate solution: while ID-Synch provides identity lifecycle management, IAM provides the real-time enforcement of strong identities by simplifying, strengthening, and tracking access to all applications.

This chapter details setup and configuration steps required for the integration of the M-Tech ID-Synch provisioning system with the Encentuate IAM access security solution. It is assumed that both ID-Synch and IMS Server have been installed.

This chapter covers the following topics:



# About M-Tech ID-Synch integration

M-Tech ID-Synch is an enterprise user provisioning (account provisioning) solution. The ID-Synch solution reduces the cost of user administration, helps new and reassigned users get to work quickly, and ensures prompt and reliable access termination.

This is accomplished through automatic propagation of changes to user profiles from systems of record to managed systems, with self-service workflow for security change requests, through consolidated and delegated user administration, and with federation. ID-Synch can manage users on over 70 types of systems.

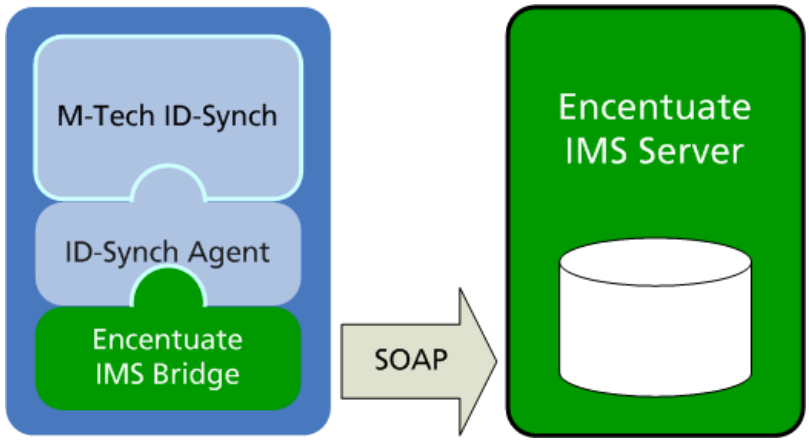
Encentuate IAM integrates with M-Tech ID-Synch to provide a complete identity and access management solution.

While ID-Synch provides the identity lifecycle management for application users, Encentuate IAM provides real-time implementation of access security policies for users and applications.

The integrated solution delivers seamless identity and access management that provides application account provisioning, a centralized view of all application accounts, sign-on/sign-off automation, authentication management, user-centric audit logs and reporting, and centralized de-provisioning of all accounts.

ID-Synch communicates with the IMS Server to populate and manage credentials in the Wallet. The Encentuate IMS Bridge and the ID-Synch-Encentuate provisioning agent are the interface engines that act as intermediaries between the IMS Server and ID-Synch.

The workflow of the provisioning process is illustrated in the following diagram:




Encentuate integration overview (M-Tech ID-Synch)

ID-Synch connects to the IMS Server via the ID-Synch-Encentuate provisioning agent and the Encentuate IMS Bridge to create IMS users, add account credentials to users' Wallets, delete account credentials from users' Wallets, and delete IMS users.

Once the ID-Synch-Encentuate provisioning agent has been loaded onto ID-Synch, all application accounts provisioned through ID-Synch will be single sign-on enabled.

Communications between the IMS Bridge and the IMS Server are done using Simple Object Access Protocol (SOAP) over HTTPS.

The following are the possible communication processes between ID-Synch and the IMS Server:

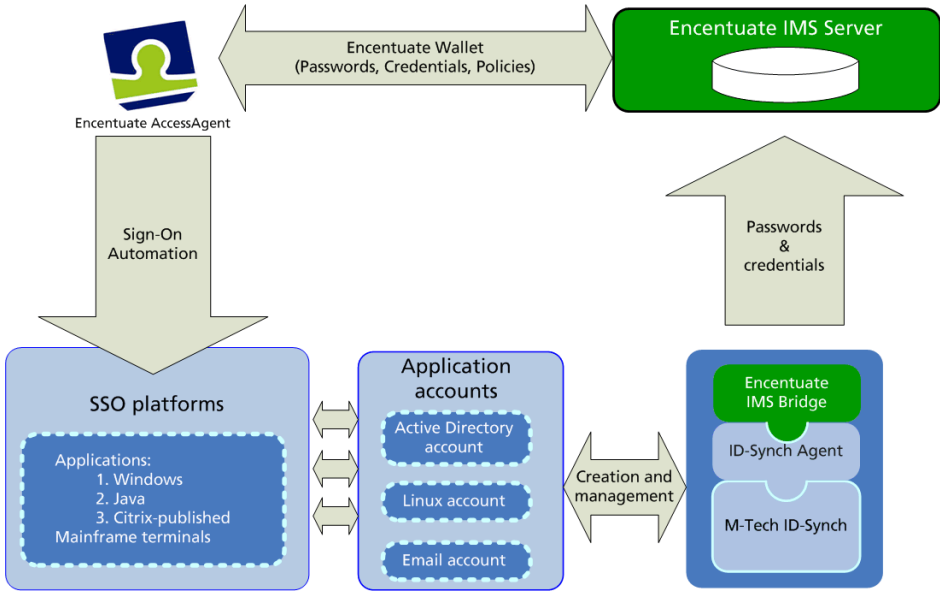
❶ When ID-Synch provisions new users, it raises an event to the ID-Synch-Encentuate provisioning agent, which communicates with the IMS Server through the IMS Bridge to create IMS users.

❷ When ID-Synch de-provisions new users, it raises an event to the ID-Synch-Encentuate provisioning agent, which communicates with the IMS Server through the IMS Bridge to delete the IMS users.

❸ When users log on to Encentuate's client software, AccessAgent, the software downloads the users' Wallets from the IMS Server and subsequently performs sign-on automation for all types of applications: enterprise, personal, certificate-enabled, and any Windows user accounts. The AccessAgent will:

- Auto-fill users' credentials into the appropriate application.
- Log the users into the application.

AccessAgent also detects any password change and synchronizes the Wallets on users' personal computers with the Wallets stored in the IMS Server.




Communication process between ID-Synch and IMS Bridge

# Minimum requirements

- M-Tech ID-Synch: 4.0
- Encentuate IMS Server: 3.0.0.0 and above
- Encentuate AccessAgent: 3.0.3.4 and above

- IMS Bridge should be deployed with JVM version 1.3.x or 1.4.x (both SUN and IBM JVM).
- For a deployment on JVM 1.3.x, two other libraries – JCE and JSSE – are provided. You need to include JCE and JSSE implementation in the path for JVM by putting necessary .JAR files in **jre/lib/ext** folder.

# Using the integration package

The ID-Synch integration package contains the following folders:

- **ProvisioningBridge [Encentuate provisioning bridge]**
  - bin
  - config [configuration files]
  - docs [documentation and notes]
    - api [Encentuate provisioning API documentation]
  - lib [Java libraries]
    - license [license text files]
- **agtencent [ID-Synch-Encentuate provisioning agent]**

*To use the integration package:*

1. Put the ProvisioningBridge folders and files (preserving folder structure) into any folder (e.g., C:\Encentuate) on the ID-Synch server.
2. Put the agtencent.exe and agtencent.jar files into the agent folder of the ID-Synch instance (e.g., **C:\Program Files\P-Synch\<INSTANCE_NAME>\agent**).

# Configuring certificate store for IMS Bridge

The IMS Bridge communicates with the IMS Server using one-way SSL. This means that the IMS Bridge needs to trust the IMS SSL certificate.

If you are deploying the IMS Bridge on an application server, where there is already one common trust store shared by different applications, you need to import the IMS SSL certificate into the key store as one trusted CA entry.

Alternatively, you need to create one key store using the Java key tool utility. Then, you need to configure the IMS Bridge to use the above trusted store.

Be sure to complete the steps in Configuring The IMS Server.

# Configuring the IMS Bridge

The IMS Bridge is packaged with a sample configuration file as below.

```
<?xml version="1.0" encoding="UTF-8"?>

 <Config>

   <main>

     <ims.serverName>

      <value xml:lang="en">ims.yourcompany.com</value>

     </ims.serverName>

     <ims.httpsPort>

      <value xml:lang="en">443</value>

     </ims.httpsPort>

   <ims.httpPort>

      <value xml:lang="en">80</value>

     </ims.httpPort>

   <ims.servicePath>

      <value xml:lang="en">/ims/services</value>

     </ims.servicePath>

   <provisioningbridge.trustStore>

<value xml:lang="en">test\config\test_keystore</value>

     </provisioningbridge.trustStore>

     <provisioningbridge.jvm.environment.initializer>

<value
xml:lang="en">encentuate.bridges.provisioning.GenericJvmEnviron
mentInitializer</value>

   </provisioningbridge.jvm.environment.initializer>

   <provisioningbridge.trustStorePassword>
```

```
      <value xml:lang="en">password</value>
      </provisioningbridge.trustStorePassword>
      <provisioningbridge.authenticationService.mapping>
       <value xml:lang="en">ActiveDirectory:dir_ad</value>
       <value xml:lang="en">LotusNotes:dir_notes</value>
      </provisioningbridge.authenticationService.mapping>
    </main>
  </Config>
```

Refer to the following XML parameters. Some of the parameters are optional.

*ims.serverName*
The DNS name of the IMS Server.

*ims.httpsPort (Optional)*
The port IMS Server listens to for HTTPS request. The default is **443**.

*ims.httpPort (Optional)*
The port IMS Server listens to for HTTP request. The default is **80**.

*ims.servicePath (Optional)*
The root path of IMS services. The default is **/ims/services/**. Note that the value should start with **/**.

*provisioningbridge.trustStore (Mandatory for CLT only)*
The trust store used by IMS Bridge (Example: C:\path\to\truststore). This configuration does not take effect if there is already one system property set for **javax.net.ssl.trustStore**.

*provisioningbridge.trustStorePassword (Mandatory for CLT only)*
Password of the trust store used by IMS Bridge. This configuration does not take effect if there is already one system property set for **javax.net.ssl.trustStorePassword**.

*provisioningbridge.password.encryption.algorithm (Optional)*
The algorithm that encrypts the provisioned application passwords. The default algorithm is **RSA/NONE/PKCS1Padding**.

*provisioningbridge.password.encryption.transformation (Optional)*
The transformation ID that corresponds to the encryption algorithm. The default is **RSA/NONE/PKCS1Padding/2048/ProvisionKeypair**.

*provisioningbridge.authenticationService.mapping (Optional)*
The mapping of application IDs on the host provisioning system to IMS Server's representation. The format of each value of this configuration key should follow the format: **prov_system_app_ID:IMS_server_app_ID**.

For example, you have configured an authentication service for Active Directory in IMS Server called **dir_encentuate.com**. However, the internal representation for the same authentication service in your provisioning system is **ENCENTUATE**.

You will then need to include the following configuration key:

```
<provisioningbridge.authenticationService.mapping>

<value xml:lang="en">ENCENTUATE:dir_encentuate.com</value>

</provisioningbridge.authenticationService.mapping>
```

*provisioningbridge.jvm.environment.initializer (Optional)*
Name of one class that implements JvmEnvironmentInitializer interface, which sets up the JVM environment such as JAVA system properties before IMS Bridge starts to run. The default is **encentuate.bridges.provisioning.GenericEnvironmentInitializer**.

# Loading ID-Synch-Encentuate provisioning agent

The ID-Synch-Encentuate provisioning agent must loaded onto the ID-Synch server.

*To load ID-Synch-Encentuate provisioning agent:*

1. Update the CLASSPATH to include the **agtencent.jar** file if the CLASSPATH is set on the host machine. If not, this step can be skipped.

2. Import the ID-Synch-Encentuate provisioning agent into ID-Synch by running the loadplatform command-line utility found in the **C:\Program Files\P-Synch\<INSTANCE_NAME>\util** folder: **loadplatform -a agtencent.exe**

# Configuring the ID-Synch Server

IDI configuration is an automated process, facilitated by placing files required by the IMS Bridge in the corresponding folder(s) and restarting the IDI.

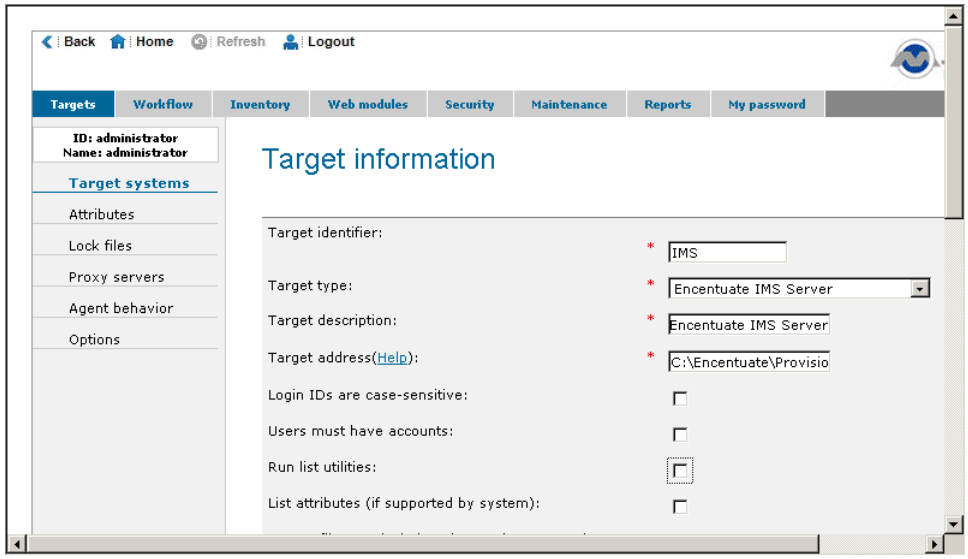
*To configure the ID-Synch Server:*

1. Log on to ID-Synch User Administration using Administrator login.

2. Create an ID-Synch target for the IMS Server by selecting *System configuration >> Targets >> Target systems >> Add*.

Note that the Address of the server should be the name of the IMS Bridge configuration file (including path, e.g., **C:\Encentuate\ProvisioningBridge\config\provisioningBridge.xml**).

Clear the **Run list utilities** checkbox, as listing is not supported by the agent.




Create ID-Synch target for IMS Server

3. ID-Synch requires that a template user account exists when creating new user accounts. In order to proceed, a dummy account needs to exist.

   Create a file in the **C:\Program Files\P-Synch\<INSTANCE_NAME>\psconfig folder called ims.lst** (where "ims" is the name given to the ID-Synch target for the IMS Server).

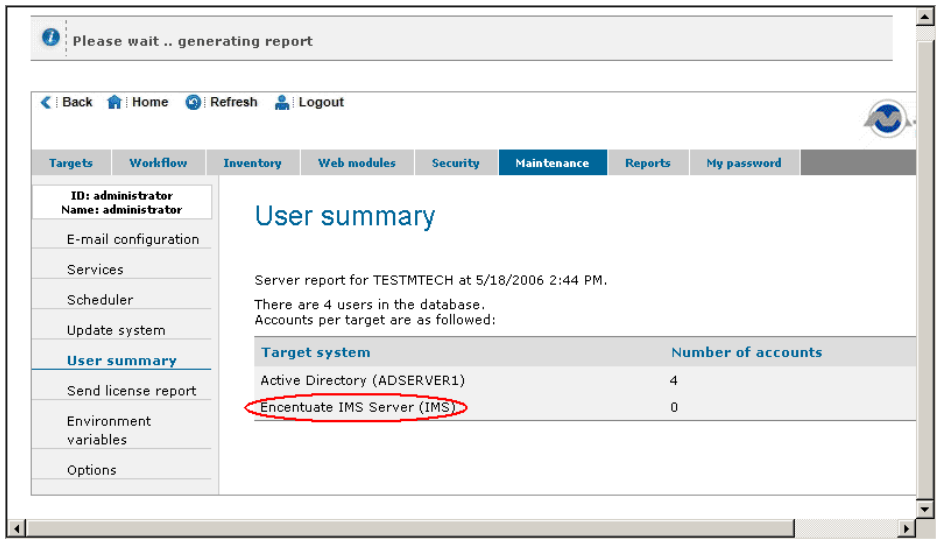
   Add the following line of text to the file: **"LONGID" "SHORTID" "FULLNAME"**

   Add another line below it providing the dummy account information. For example: **"template" "template" "Template Account"**

   Save the file and close it.

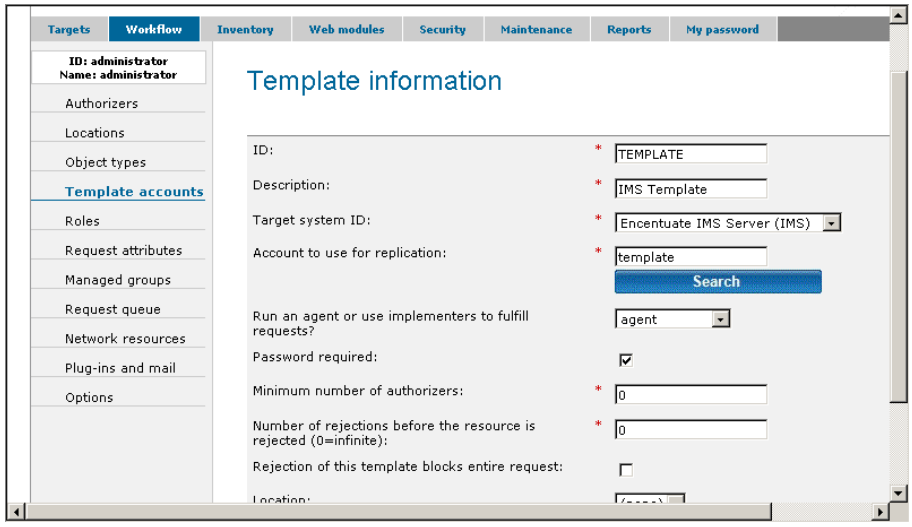
4. Run psupdate from the command line or select *Maintenance >> Update system >> Update now*.

5. Verify that the user is found by checking *Maintenance >> User summary*.




Checking user summary

❻ Set the template account for the IMS target by selecting *Workflow >> Template accounts*.

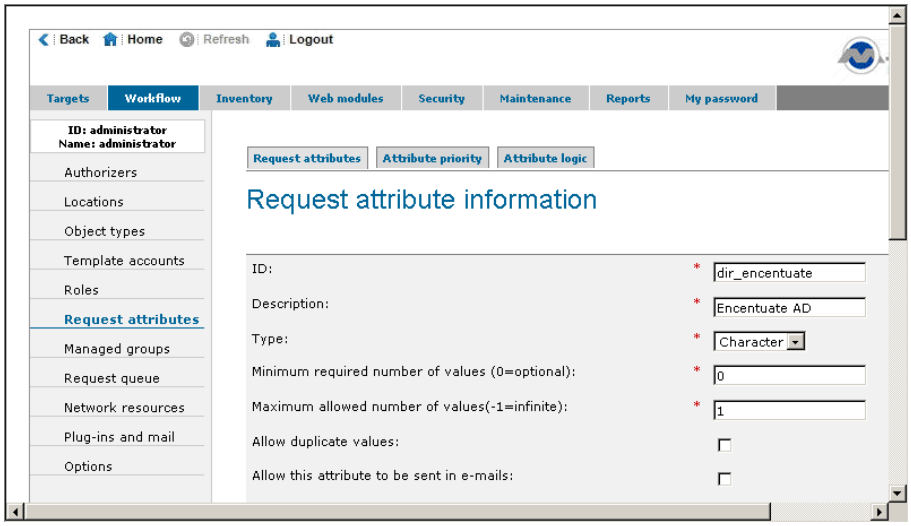



Set template account for IMS target

❼ Add the IMS authentication services (Wallet applications) as Request Attributes by selecting *Workflow >> Request attributes >> Request attributes*.

The attribute ID should be the authentication service ID, and an appropriate description entered.

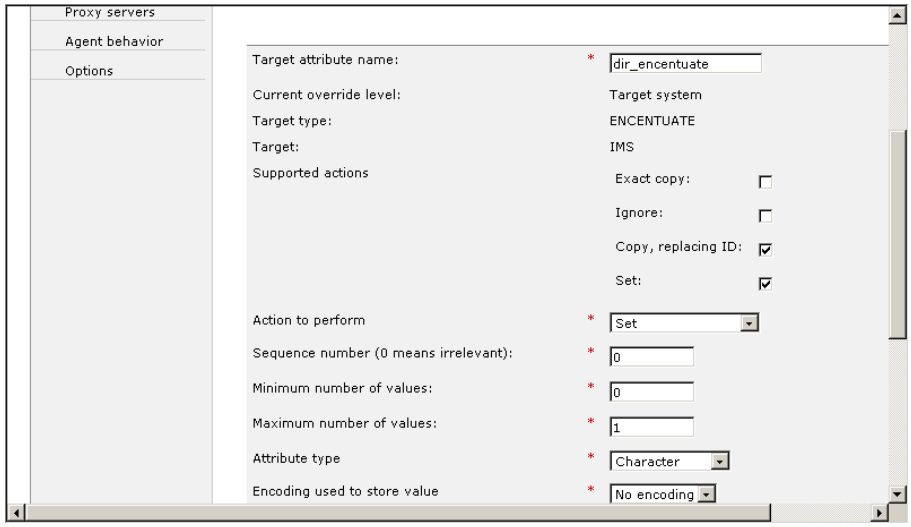
As IMS allows multiple application instances to exist in a Wallet (provided the user names are different), the **Maximum allowed number of values** should be set to the appropriate number. The information entered into the **Description** field (instead of the attribute ID) is displayed to users in ID-Synch.



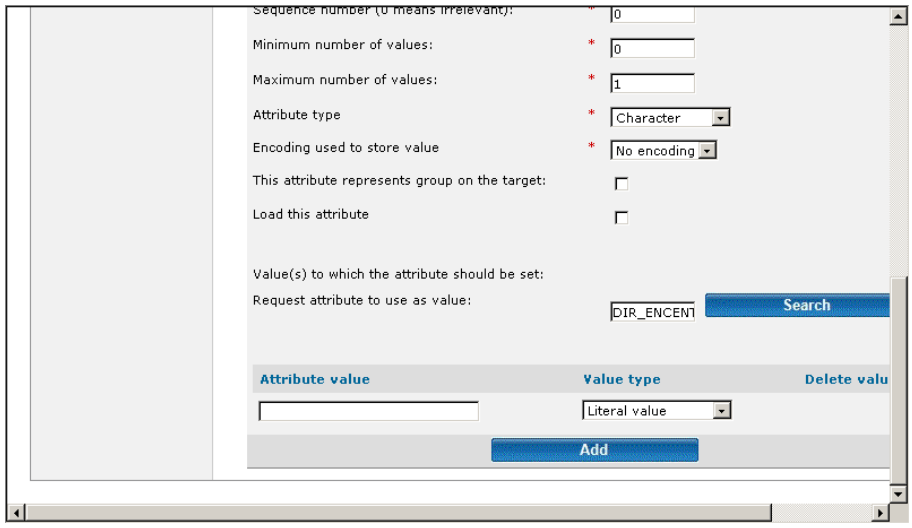

Add authentication services as request attributes

8 Add the IMS authentication services as Target Attributes by selecting *Targets >> Attributes >> Target >> Select button (Encentuate IMS Server) >> Add*.

The target attribute name should be the authentication service ID, and the supported actions are **Copy, replacing ID** and **Set**, with **Set** being the action to perform. The maximum number of values should be the same as that set for the request attributes, attribute type should be **Character**, and the request attribute should correspond to the appropriate one created in the previous step.

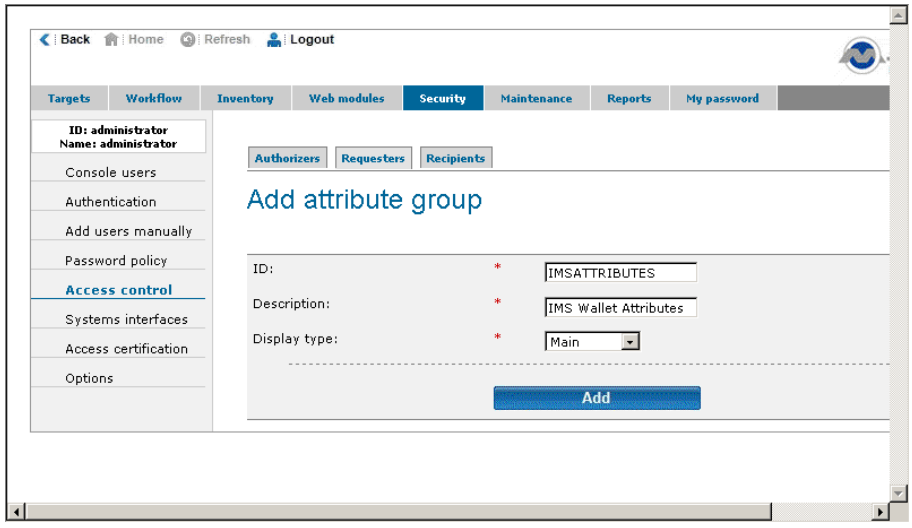



Add target attributes (1)



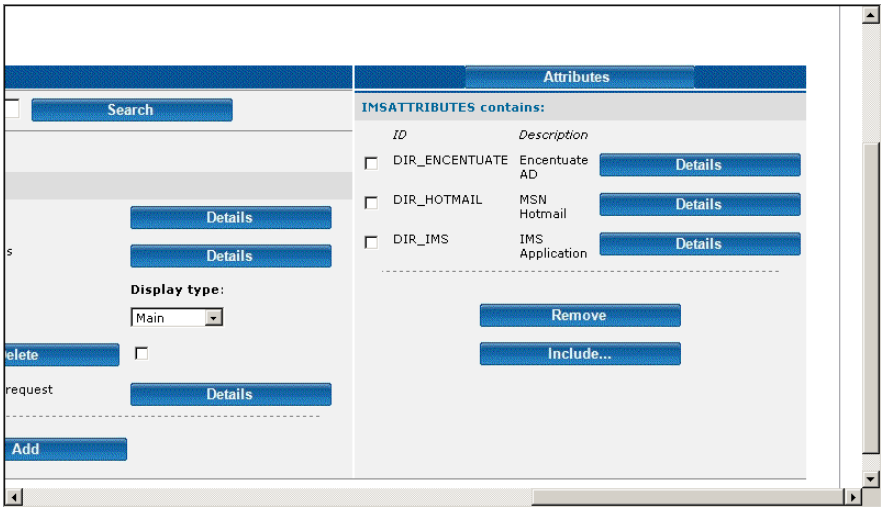

Add target attributes (2)

❾ Add the appropriate requester security access for the IMS attributes by selecting *Security >> Access control >> Requesters >> Attribute groups*. Click the **Add** button to add the IMSATTRIBUTES group.

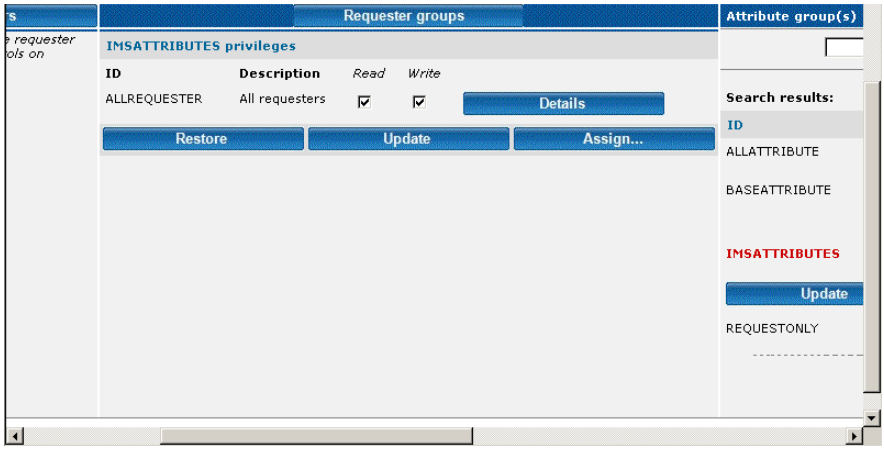



Add IMSATTRIBUTES group for requester security access

❿ Add the appropriate IMS attributes to the IMSATTRIBUTES group by clicking **Details**. For each attribute, click **Include...** button. Finally, assign the privileges to the requester group (Read and Write).




Add attributes to IMSATTRIBUTES group




Assign privileges to requester group

⓫ Go to *Home >> User administration >> New user profile* to add a user. This should complete the process for creating a user with applications accounts populated in the Wallet.

Creating Request Attributes for the user by adding application accounts to the Wallet. Actual provisioning of application accounts on the application servers are done by adding Resources, which are two separate processes.

*The provisioning of application accounts can only be done during user creation.*

# Provisioning setup and maintenance

*To provision a new user (M-Tech ID-Synch):*

❶ ID-Synch provides the IMS user names and initial passwords to ID-Synch-Encentuate provisioning agent, which in turn will provision the IMS users through the IMS Bridge. The password can be randomly-generated, or manually assigned through ID-Synch.

❷ The ID-Synch-Encentuate provisioning agent connects to the IMS Bridge to provision the IMS users using the IMS user name and password generated by ID-Synch. The users' Wallets will also be initialized on the IMS Server at this time such that account credentials can then be added to the Wallets.

*IMS user accounts should be created before other application accounts. Account credentials cannot be added to the user's Wallet before the users are provisioned on the IMS Server.*

❸ The users log on with their IMS user names and initial passwords when they use AccessAgent for the first time. They will be prompted to change the initial password and the Wallet containing the provisioned account credentials will be downloaded from the IMS Server.

*To de-provision users (M-Tech ID-Synch):*

❶ ID-Synch provides the IMS user ID of the users to be de-provisioned to ID-Synch-Encentuate provisioning agent.

❷ The ID-Synch-Encentuate provisioning agent connects to the IMS Bridge and revokes the IMS users. The revocation of the IMS users will invalidate both the users' accounts and the users' Wallets on the server.

❸ If the users attempt to log on using AccessAgent, the log on will fail and Wallets that have been cached locally by AccessAgent will be revoked.

# APPENDICES

# Appendices

Refer to the following appendices for more useful information on integrating provisioning solutions with Encentuate IAM:

- Appendix A: Configuring The IMS Server
- Appendix B: WSDL for Server Authentication
- Appendix C: WSDL for Provisioning Service
- Appendix D: Troubleshooting
- Appendix e: Keytool

# Configuring The IMS Server

*To configure the IMS Server:*

1. Use the IMS Configuration Utility (Authentication Services section) to add the authentication services of the applications to be provisioned.

2. Using AccessAdmin (under Authentication service policies section), configure each authentication service to use the appropriate authentication modes (e.g., Password).

3. The provisioning agent needs to authenticate with the IMS Server before it can call the provisioning services. This authentication is done through using a shared secret between the provisioning agent and the IMS Server.

   Use the IMS Configuration Utility (IMS Bridges section) to configure these settings. Alternatively, the shared secret can be configured in the IMS configuration file **<IMS Installation Folder>\ims\config\ims.xml** with the following entries:

```
<auth.server.agent.clientIp>

   <value xml:lang="en">10.1.16.60</value>

</auth.server.agent.clientIp>



<auth.server.agent.password>

   <value xml:lang="en">sharedsecret</value>

</auth.server.agent.password>
```

The above configuration means that the provisioning agent with server ID **agent** is deployed on a machine with IP address **10.1.16.60**. It is allowed to log on to the IMS Server with server ID **agent** and password **sharedsecret**.

*Restart the IMS Server after completing the configuration. When the IMS Server starts, it encrypts the password and replaces the clear text password with the encrypted password in the configuration file.*

# WSDL for Server Authentication

The WSDL for the server authentication API can be obtained from an installed IMS Server at the following URL (**imsserver** should be replaced by the hostname of your IMS Server): https://imsserver/ims/services/encentuate.ims.service.ServerAuthentication?wsdl

It is also reproduced here for reference:

```
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitions targetNamespace="https://imsserver/ims/
services/encentuate.ims.service.ServerAuthentication"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="https://imsserver/ims/services/
encentuate.ims.service.ServerAuthentication"
xmlns:intf="https://imsserver/ims/services/
encentuate.ims.service.ServerAuthentication"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="http://result.ims.encentuate" xmlns:wsdl="http://
schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://
schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/
2001/XMLSchema"><wsdl:types><schema targetNamespace="http://
result.ims.encentuate" xmlns="http://www.w3.org/2001/
XMLSchema"><import namespace="http://schemas.xmlsoap.org/soap/
encoding/"/><complexType
name="ResultMessage"><sequence><element name="resultCode"
type="xsd:int"/><element name="resultString" nillable="true"
type="xsd:string"/></sequence></complexType><element
name="ResultMessage" nillable="true" type="tns1:ResultMessage"/
></schema></wsdl:types>

 <wsdl:message name="loginByPasswordResponse">

 <wsdl:part name="loginByPasswordReturn"
type="tns1:ResultMessage"/>

 </wsdl:message>

 <wsdl:message name="terminateSessionResponse">

 <wsdl:part name="terminateSessionReturn" type="xsd:int"/>
```

```
 </wsdl:message>

 <wsdl:message name="loginByPasswordRequest">

 <wsdl:part name="serverId" type="xsd:string"/>

 <wsdl:part name="password" type="xsd:string"/>

 </wsdl:message>

 <wsdl:message name="terminateSessionRequest">

 <wsdl:part name="sessionKey" type="xsd:string"/>

 </wsdl:message>

 <wsdl:portType name="ServerAuthentication">

 <wsdl:operation name="terminateSession"
parameterOrder="sessionKey">

 <wsdl:input message="impl:terminateSessionRequest"
name="terminateSessionRequest"/>

 <wsdl:output message="impl:terminateSessionResponse"
name="terminateSessionResponse"/>

 </wsdl:operation>

 <wsdl:operation name="loginByPassword"
parameterOrder="serverId password">

 <wsdl:input message="impl:loginByPasswordRequest"
name="loginByPasswordRequest"/>

 <wsdl:output message="impl:loginByPasswordResponse"
name="loginByPasswordResponse"/>

 </wsdl:operation>

 </wsdl:portType>

 <wsdl:binding
name="encentuate.ims.service.ServerAuthenticationSoapBinding"
type="impl:ServerAuthentication">

 <wsdlsoap:binding style="rpc" transport="http://
schemas.xmlsoap.org/soap/http"/>

 <wsdl:operation name="terminateSession">

 <wsdlsoap:operation soapAction="soapAction=""/>

 <wsdl:input name="terminateSessionRequest">

 <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/" namespace="https://imsserver/ims/services/
encentuate.ims.service.ServerAuthentication" use="encoded"/>
```

```
 </wsdl:input>

 <wsdl:output name="terminateSessionResponse">

 <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/" namespace="https://imsserver/ims/services/
encentuate.ims.service.ServerAuthentication" use="encoded"/>

 </wsdl:output>

 </wsdl:operation>

 <wsdl:operation name="loginByPassword">

 <wsdlsoap:operation soapAction="soapAction=""/>

 <wsdl:input name="loginByPasswordRequest">

 <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/" namespace="https://imsserver/ims/services/
encentuate.ims.service.ServerAuthentication" use="encoded"/>

 </wsdl:input>

 <wsdl:output name="loginByPasswordResponse">

 <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/" namespace="https://imsserver/ims/services/
encentuate.ims.service.ServerAuthentication" use="encoded"/>

 </wsdl:output>

 </wsdl:operation>

 </wsdl:binding>

 <wsdl:service name="ServerAuthenticationService">

 <wsdl:port
binding="impl:encentuate.ims.service.ServerAuthenticationSoapBi
nding" name="encentuate.ims.service.ServerAuthentication">

 <wsdlsoap:address location="https://imsserver/ims/services/
encentuate.ims.service.ServerAuthentication"/>

 </wsdl:port>

 </wsdl:service>

</wsdl:definitions>
```

# WSDL for Provisioning Service

The WSDL for the provisioning service API can be obtained from an installed IMS Server at the following URL (**imsserver** should be replaced by the hostname of your IMS Server): https://imsserver/ims/services/encentuate.ims.service.ProvisioningService?wsdl

It is also reproduced here for reference:

```
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitions targetNamespace=""https://imsserver/ims/
services/encentuate.ims.service.ProvisioningService"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="https://imsserver/ims/services/
encentuate.ims.service.ProvisioningService" xmlns:intf="https:/
/imsserver/ims/services/
encentuate.ims.service.ProvisioningService"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="http://result.ims.encentuate" xmlns:wsdl="http://
schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://
schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/
2001/XMLSchema"><wsdl:types><schema targetNamespace="http://
result.ims.encentuate" xmlns="http://www.w3.org/2001/
XMLSchema"><import namespace="http://schemas.xmlsoap.org/soap/
encoding/"/><complexType name="NameValue"><sequence><element
name="name" nillable="true" type="xsd:string"/><element
name="value" nillable="true" type="xsd:string"/></sequence></
complexType><complexType
name="ResultMessage"><sequence><element name="resultCode"
type="xsd:int"/><element name="resultString" nillable="true"
type="xsd:string"/></sequence></complexType><element
name="ResultMessage" nillable="true" type="tns1:ResultMessage"/
><complexType name="ResultArrayMap"><sequence><element
name="maps" nillable="true" type="impl:ArrayOf_apachesoap_Map"/
><element name="resultCode" type="xsd:int"/></sequence></
complexType><element name="ResultArrayMap" nillable="true"
type="tns1:ResultArrayMap"/></schema><schema
targetNamespace="https://imsserver/ims/services/
encentuate.ims.service.ProvisioningService" xmlns="http://
www.w3.org/2001/XMLSchema"><import namespace="http://
schemas.xmlsoap.org/soap/encoding/"/><complexType
```

```
name="ArrayOf_tns1_NameValue"><complexContent><restriction
base="soapenc:Array"><attribute ref="soapenc:arrayType"
wsdl:arrayType="tns1:NameValue[]"/></restriction></
complexContent></complexType><element
name="ArrayOf_tns1_NameValue" nillable="true"
type="impl:ArrayOf_tns1_NameValue"/><complexType
name="ArrayOf_apachesoap_Map"><complexContent><restriction
base="soapenc:Array"><attribute ref="soapenc:arrayType"
wsdl:arrayType="apachesoap:Map[]"/></restriction></
complexContent></complexType></schema><schema
targetNamespace="http://xml.apache.org/xml-soap" xmlns="http://
www.w3.org/2001/XMLSchema"><import namespace="http://
schemas.xmlsoap.org/soap/encoding/"/><complexType
name="Map"><sequence><element maxOccurs="unbounded"
minOccurs="0" name="item"><complexType><all><element name="key"
type="xsd:anyType"/><element name="value" type="xsd:anyType"/
></all></complexType></element></sequence></complexType></
schema></wsdl:types>

 <wsdl:message name="revokeImsAccountUsingAttrsRequest">

    <wsdl:part name="sessionId" type="xsd:string"/>

    <wsdl:part name="attributes"
type="impl:ArrayOf_tns1_NameValue"/>

 </wsdl:message>

 <wsdl:message name="preProvisionImsUserWithEntIdResponse">

    <wsdl:part name="preProvisionImsUserWithEntIdReturn"
type="xsd:int"/>

 </wsdl:message>

 <wsdl:message name="createWalletResponse">

    <wsdl:part name="createWalletReturn"
type="tns1:ResultMessage"/>

 </wsdl:message>

 <wsdl:message name="setAccountCredentialResponse">

    <wsdl:part name="setAccountCredentialReturn"
type="xsd:int"/>

 </wsdl:message>

 <wsdl:message name="deleteImsAccountResponse">

    <wsdl:part name="deleteImsAccountReturn" type="xsd:int"/>

 </wsdl:message>

 <wsdl:message name="revokeImsAccountResponse">

    <wsdl:part name="revokeImsAccountReturn" type="xsd:int"/>
```

```
</wsdl:message>
<wsdl:message name="createWalletRequest">
   <wsdl:part name="sessionId" type="xsd:string"/>
   <wsdl:part name="enterpriseId" type="xsd:string"/>
   <wsdl:part name="walletId" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="removeAccountCredentialWithEntIdRequest">
   <wsdl:part name="sessionId" type="xsd:string"/>
   <wsdl:part name="adminEntId" type="xsd:string"/>
   <wsdl:part name="enterpriseId" type="xsd:string"/>
   <wsdl:part name="authId" type="xsd:string"/>
   <wsdl:part name="accountType" type="xsd:string"/>
   <wsdl:part name="username" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="removeAccountCredentialRequest">
   <wsdl:part name="sessionId" type="xsd:string"/>
   <wsdl:part name="enterpriseId" type="xsd:string"/>
   <wsdl:part name="authId" type="xsd:string"/>
   <wsdl:part name="accountType" type="xsd:string"/>
   <wsdl:part name="username" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="setAccountCredentialRequest">
   <wsdl:part name="sessionId" type="xsd:string"/>
   <wsdl:part name="enterpriseId" type="xsd:string"/>
   <wsdl:part name="authId" type="xsd:string"/>
   <wsdl:part name="accountType" type="xsd:string"/>
   <wsdl:part name="username" type="xsd:string"/>
   <wsdl:part name="password" type="xsd:string"/>
</wsdl:message>
```

```
 <wsdl:message name="revokeImsAccountUsingAttrsResponse">

    <wsdl:part name="revokeImsAccountUsingAttrsReturn"
type="xsd:int"/>

 </wsdl:message>

 <wsdl:message name="revokeImsAccountWithEntIdResponse">

    <wsdl:part name="revokeImsAccountWithEntIdReturn"
type="xsd:int"/>

 </wsdl:message>

 <wsdl:message name="getProvisioningCertResponse">

    <wsdl:part name="getProvisioningCertReturn"
type="tns1:ResultMessage"/>

 </wsdl:message>

 <wsdl:message name="getUserAccountsRequest">

    <wsdl:part name="sessionId" type="xsd:string"/>

    <wsdl:part name="enterpriseId" type="xsd:string"/>

 </wsdl:message>

 <wsdl:message name="deleteImsAccountRequest">

    <wsdl:part name="sessionId" type="xsd:string"/>

    <wsdl:part name="enterpriseId" type="xsd:string"/>

 </wsdl:message>

 <wsdl:message name="deleteImsAccountUsingAttrsRequest">

    <wsdl:part name="sessionId" type="xsd:string"/>

    <wsdl:part name="attributes"
type="impl:ArrayOf_tns1_NameValue"/>

 </wsdl:message>

 <wsdl:message name="removeAccountCredentialResponse">

    <wsdl:part name="removeAccountCredentialReturn"
type="xsd:int"/>

 </wsdl:message>

 <wsdl:message name="addAccountCredentialWithEntIdResponse">

    <wsdl:part name="addAccountCredentialWithEntIdReturn"
type="xsd:int"/>
```

```
 </wsdl:message>

 <wsdl:message name="removeAccountCredentialWithEntIdResponse">

    <wsdl:part name="removeAccountCredentialWithEntIdReturn"
type="xsd:int"/>

 </wsdl:message>

 <wsdl:message name="deleteImsAccountWithEntIdResponse">

    <wsdl:part name="deleteImsAccountWithEntIdReturn"
type="xsd:int"/>

 </wsdl:message>

 <wsdl:message name="getUserAccountsResponse">

    <wsdl:part name="getUserAccountsReturn"
type="tns1:ResultArrayMap"/>

 </wsdl:message>

 <wsdl:message name="preProvisionImsUserRequest">

    <wsdl:part name="sessionId" type="xsd:string"/>

    <wsdl:part name="enterpriseId" type="xsd:string"/>

    <wsdl:part name="initialPassword" type="xsd:string"/>

    <wsdl:part name="attributes"
type="impl:ArrayOf_tns1_NameValue"/>

 </wsdl:message>

 <wsdl:message name="setAccountCredentialWithEntIdRequest">

    <wsdl:part name="sessionId" type="xsd:string"/>

    <wsdl:part name="enterpriseId" type="xsd:string"/>

    <wsdl:part name="adminEntId" type="xsd:string"/>

    <wsdl:part name="authId" type="xsd:string"/>

    <wsdl:part name="accountType" type="xsd:string"/>

    <wsdl:part name="username" type="xsd:string"/>

    <wsdl:part name="password" type="xsd:string"/>

 </wsdl:message>

 <wsdl:message name="preProvisionImsUserWithEntIdRequest">

    <wsdl:part name="sessionId" type="xsd:string"/>
```

```
    <wsdl:part name="enterpriseId" type="xsd:string"/>

    <wsdl:part name="adminEntId" type="xsd:string"/>

    <wsdl:part name="initialPassword" type="xsd:string"/>

    <wsdl:part name="attributes"
type="impl:ArrayOf_tns1_NameValue"/>

 </wsdl:message>

 <wsdl:message name="revokeImsAccountWithEntIdRequest">

    <wsdl:part name="sessionId" type="xsd:string"/>

    <wsdl:part name="enterpriseId" type="xsd:string"/>

    <wsdl:part name="adminEntId" type="xsd:string"/>

 </wsdl:message>

 <wsdl:message name="revokeImsAccountRequest">

    <wsdl:part name="sessionId" type="xsd:string"/>

    <wsdl:part name="enterpriseId" type="xsd:string"/>

 </wsdl:message>

 <wsdl:message name="getRegistrationStatusResponse">

    <wsdl:part name="getRegistrationStatusReturn"
type="xsd:int"/>

 </wsdl:message>

 <wsdl:message name="deleteImsAccountWithEntIdRequest">

    <wsdl:part name="sessionId" type="xsd:string"/>

    <wsdl:part name="enterpriseId" type="xsd:string"/>

    <wsdl:part name="adminEntId" type="xsd:string"/>

 </wsdl:message>

 <wsdl:message name="getProvisioningCertRequest">

    <wsdl:part name="sessionId" type="xsd:string"/>

    <wsdl:part name="enterpriseId" type="xsd:string"/>

 </wsdl:message>

 <wsdl:message name="addAccountCredentialWithEntIdRequest">

    <wsdl:part name="sessionId" type="xsd:string"/>
```

```
      <wsdl:part name="enterpriseId" type="xsd:string"/>

      <wsdl:part name="adminEntId" type="xsd:string"/>

      <wsdl:part name="authId" type="xsd:string"/>

      <wsdl:part name="accountType" type="xsd:string"/>

      <wsdl:part name="username" type="xsd:string"/>

      <wsdl:part name="password" type="xsd:string"/>

   </wsdl:message>

   <wsdl:message name="addAccountCredentialResponse">

      <wsdl:part name="addAccountCredentialReturn"
  type="xsd:int"/>

   </wsdl:message>

   <wsdl:message name="getRegistrationStatusRequest">

      <wsdl:part name="sessionId" type="xsd:string"/>

      <wsdl:part name="enterpriseId" type="xsd:string"/>

   </wsdl:message>

   <wsdl:message name="deleteImsAccountUsingAttrsResponse">

      <wsdl:part name="deleteImsAccountUsingAttrsReturn"
  type="xsd:int"/>

   </wsdl:message>

   <wsdl:message name="preProvisionImsUserResponse">

      <wsdl:part name="preProvisionImsUserReturn" type="xsd:int"/
  >

   </wsdl:message>

   <wsdl:message name="addAccountCredentialRequest">

      <wsdl:part name="sessionId" type="xsd:string"/>

      <wsdl:part name="enterpriseId" type="xsd:string"/>

      <wsdl:part name="authId" type="xsd:string"/>

      <wsdl:part name="accountType" type="xsd:string"/>

      <wsdl:part name="username" type="xsd:string"/>

      <wsdl:part name="password" type="xsd:string"/>

   </wsdl:message>
```

```
 <wsdl:message name="setAccountCredentialWithEntIdResponse">

    <wsdl:part name="setAccountCredentialWithEntIdReturn"
type="xsd:int"/>

 </wsdl:message>

 <wsdl:portType name="ProvisioningService">

    <wsdl:operation name="getRegistrationStatus"
parameterOrder="sessionId enterpriseId">

      <wsdl:input message="impl:getRegistrationStatusRequest"
name="getRegistrationStatusRequest"/>

     <wsdl:output message="impl:getRegistrationStatusResponse"
name="getRegistrationStatusResponse"/>

    </wsdl:operation>

    <wsdl:operation name="preProvisionImsUser"
parameterOrder="sessionId enterpriseId initialPassword
attributes">

      <wsdl:input message="impl:preProvisionImsUserRequest"
name="preProvisionImsUserRequest"/>

      <wsdl:output message="impl:preProvisionImsUserResponse"
name="preProvisionImsUserResponse"/>

    </wsdl:operation>

    <wsdl:operation name="preProvisionImsUserWithEntId"
parameterOrder="sessionId enterpriseId adminEntId
initialPassword attributes">

      <wsdl:input
message="impl:preProvisionImsUserWithEntIdRequest"
name="preProvisionImsUserWithEntIdRequest"/>

      <wsdl:output
message="impl:preProvisionImsUserWithEntIdResponse"
name="preProvisionImsUserWithEntIdResponse"/>

    </wsdl:operation>

    <wsdl:operation name="createWallet"
parameterOrder="sessionId enterpriseId walletId">

      <wsdl:input message="impl:createWalletRequest"
name="createWalletRequest"/>

      <wsdl:output message="impl:createWalletResponse"
name="createWalletResponse"/>

    </wsdl:operation>
```

```
    <wsdl:operation name="getProvisioningCert"
parameterOrder="sessionId enterpriseId">

      <wsdl:input message="impl:getProvisioningCertRequest"
name="getProvisioningCertRequest"/>

      <wsdl:output message="impl:getProvisioningCertResponse"
name="getProvisioningCertResponse"/>

    </wsdl:operation>

    <wsdl:operation name="addAccountCredential"
parameterOrder="sessionId enterpriseId authId accountType
username password">

      <wsdl:input message="impl:addAccountCredentialRequest"
name="addAccountCredentialRequest"/>

      <wsdl:output message="impl:addAccountCredentialResponse"
name="addAccountCredentialResponse"/>

    </wsdl:operation>

    <wsdl:operation name="addAccountCredentialWithEntId"
parameterOrder="sessionId enterpriseId adminEntId authId
accountType username password">

      <wsdl:input
message="impl:addAccountCredentialWithEntIdRequest"
name="addAccountCredentialWithEntIdRequest"/>

      <wsdl:output
message="impl:addAccountCredentialWithEntIdResponse"
name="addAccountCredentialWithEntIdResponse"/>

    </wsdl:operation>

    <wsdl:operation name="removeAccountCredential"
parameterOrder="sessionId enterpriseId authId accountType
username">

     <wsdl:input message="impl:removeAccountCredentialRequest"
name="removeAccountCredentialRequest"/>

      <wsdl:output
message="impl:removeAccountCredentialResponse"
name="removeAccountCredentialResponse"/>

    </wsdl:operation>

    <wsdl:operation name="removeAccountCredentialWithEntId"
parameterOrder="sessionId adminEntId enterpriseId authId
accountType username">

      <wsdl:input
message="impl:removeAccountCredentialWithEntIdRequest"
name="removeAccountCredentialWithEntIdRequest"/>
```

```
      <wsdl:output
message="impl:removeAccountCredentialWithEntIdResponse"
name="removeAccountCredentialWithEntIdResponse"/>

    </wsdl:operation>

    <wsdl:operation name="setAccountCredential"
parameterOrder="sessionId enterpriseId authId accountType
username password">

      <wsdl:input message="impl:setAccountCredentialRequest"
name="setAccountCredentialRequest"/>

      <wsdl:output message="impl:setAccountCredentialResponse"
name="setAccountCredentialResponse"/>

    </wsdl:operation>

    <wsdl:operation name="setAccountCredentialWithEntId"
parameterOrder="sessionId enterpriseId adminEntId authId
accountType username password">

      <wsdl:input
message="impl:setAccountCredentialWithEntIdRequest"
name="setAccountCredentialWithEntIdRequest"/>

      <wsdl:output
message="impl:setAccountCredentialWithEntIdResponse"
name="setAccountCredentialWithEntIdResponse"/>

    </wsdl:operation>

    <wsdl:operation name="revokeImsAccountUsingAttrs"
parameterOrder="sessionId attributes">

      <wsdl:input
message="impl:revokeImsAccountUsingAttrsRequest"
name="revokeImsAccountUsingAttrsRequest"/>

      <wsdl:output
message="impl:revokeImsAccountUsingAttrsResponse"
name="revokeImsAccountUsingAttrsResponse"/>

    </wsdl:operation>

    <wsdl:operation name="deleteImsAccountUsingAttrs"
parameterOrder="sessionId attributes">

      <wsdl:input
message="impl:deleteImsAccountUsingAttrsRequest"
name="deleteImsAccountUsingAttrsRequest"/>

      <wsdl:output
message="impl:deleteImsAccountUsingAttrsResponse"
name="deleteImsAccountUsingAttrsResponse"/>

    </wsdl:operation>
```

```
    <wsdl:operation name="revokeImsAccount"
parameterOrder="sessionId enterpriseId">

      <wsdl:input message="impl:revokeImsAccountRequest"
name="revokeImsAccountRequest"/>

      <wsdl:output message="impl:revokeImsAccountResponse"
name="revokeImsAccountResponse"/>

    </wsdl:operation>

    <wsdl:operation name="revokeImsAccountWithEntId"
parameterOrder="sessionId enterpriseId adminEntId">

      <wsdl:input
message="impl:revokeImsAccountWithEntIdRequest"
name="revokeImsAccountWithEntIdRequest"/>

      <wsdl:output
message="impl:revokeImsAccountWithEntIdResponse"
name="revokeImsAccountWithEntIdResponse"/>

    </wsdl:operation>

    <wsdl:operation name="deleteImsAccount"
parameterOrder="sessionId enterpriseId">

      <wsdl:input message="impl:deleteImsAccountRequest"
name="deleteImsAccountRequest"/>

      <wsdl:output message="impl:deleteImsAccountResponse"
name="deleteImsAccountResponse"/>

    </wsdl:operation>

    <wsdl:operation name="deleteImsAccountWithEntId"
parameterOrder="sessionId enterpriseId adminEntId">

      <wsdl:input
message="impl:deleteImsAccountWithEntIdRequest"
name="deleteImsAccountWithEntIdRequest"/>

      <wsdl:output
message="impl:deleteImsAccountWithEntIdResponse"
name="deleteImsAccountWithEntIdResponse"/>

    </wsdl:operation>

    <wsdl:operation name="getUserAccounts"
parameterOrder="sessionId enterpriseId">

      <wsdl:input message="impl:getUserAccountsRequest"
name="getUserAccountsRequest"/>

      <wsdl:output message="impl:getUserAccountsResponse"
name="getUserAccountsResponse"/>

    </wsdl:operation>
```

```
 </wsdl:portType>

 <wsdl:binding
name="encentuate.ims.service.ProvisioningServiceSoapBinding"
type="impl:ProvisioningService">

    <wsdlsoap:binding style="rpc" transport="http://
schemas.xmlsoap.org/soap/http"/>

    <wsdl:operation name="getRegistrationStatus">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="getRegistrationStatusRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="getRegistrationStatusResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="preProvisionImsUser">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="preProvisionImsUserRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="preProvisionImsUserResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="preProvisionImsUserWithEntId">
```

```
      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="preProvisionImsUserWithEntIdRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="preProvisionImsUserWithEntIdResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="createWallet">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="createWalletRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="createWalletResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="getProvisioningCert">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="getProvisioningCertRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>
```

```
      <wsdl:output name="getProvisioningCertResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="addAccountCredential">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="addAccountCredentialRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="addAccountCredentialResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="addAccountCredentialWithEntId">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="addAccountCredentialWithEntIdRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output
name="addAccountCredentialWithEntIdResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>
```

```
    </wsdl:operation>

    <wsdl:operation name="removeAccountCredential">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="removeAccountCredentialRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="removeAccountCredentialResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="removeAccountCredentialWithEntId">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input
name="removeAccountCredentialWithEntIdRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output
name="removeAccountCredentialWithEntIdResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="setAccountCredential">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="setAccountCredentialRequest">
```

```
        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="setAccountCredentialResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="setAccountCredentialWithEntId">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="setAccountCredentialWithEntIdRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output
name="setAccountCredentialWithEntIdResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="revokeImsAccountUsingAttrs">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="revokeImsAccountUsingAttrsRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="revokeImsAccountUsingAttrsResponse">
```

```
        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="deleteImsAccountUsingAttrs">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="deleteImsAccountUsingAttrsRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="deleteImsAccountUsingAttrsResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="revokeImsAccount">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="revokeImsAccountRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="revokeImsAccountResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="revokeImsAccountWithEntId">
```

```
      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="revokeImsAccountWithEntIdRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="revokeImsAccountWithEntIdResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="deleteImsAccount">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="deleteImsAccountRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="deleteImsAccountResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="deleteImsAccountWithEntId">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="deleteImsAccountWithEntIdRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>
```

```
      <wsdl:output name="deleteImsAccountWithEntIdResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="getUserAccounts">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="getUserAccountsRequest">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="getUserAccountsResponse">

        <wsdlsoap:body encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/" namespace="https://
imsserver/ims/services/
encentuate.ims.service.ProvisioningService" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

 </wsdl:binding>

 <wsdl:service name="ProvisioningServiceService">

    <wsdl:port
binding="impl:encentuate.ims.service.ProvisioningServiceSoapBin
ding" name="encentuate.ims.service.ProvisioningService">

      <wsdlsoap:address location="https://imsserver/ims/
services/encentuate.ims.service.ProvisioningService"/>

    </wsdl:port>

 </wsdl:service>

</wsdl:definitions>
```

# Troubleshooting

Refer to the following topics to find a solution to specific issues:



## Logs for ITIM and WebSphere Application Server

The ITIM 4.6 log file path is specified in **enRoleLogging.properties** located in **<ITIM_HOME>\data** directory. The default file name is trace.log.

For ITIM 4.5, the ITIM logs are at **<WEBSPHERE_HOME>\logs\itim.log**.

WebSphere logs can be found at
**<WEBSPHERE_HOME>\logs\server1\SystemOut.log** and
**<WEBSPHERE_HOME>\logs\server1\SystemErr.log**.

## Exceptions like ClassNotFoundException and LinkageError

If you deploy the provisioning bridge in an application server such as Weblogic or Websphere, you may get errors like "ClassNotFoundException because of conflicts between libraries shipped with the provisioning bridge and the host application server".

To understand such issues, refer to documentation about class loader mechanisms in the application server. Then, search in library folders used by different class loaders to determine those conflicted libraries.

If possible, do not share libraries between the provisioning bridge and other modules. This can be done by putting the provisioning bridge into a separate EAR module, etc.

Otherwise, attempt to resolve the library conflicts by removing libraries in the provisioning bridge (i.e., try to use the existing library in the application server). However, this may not work because this provisioning bridge may require libraries with specific versions.

## Exceptions like javax.net.ssl.SSLHandshakeException or java.net.SocketException

Check if you pointed the provisioning bridge to the correct configuration file and check your configuration file to determine if the information about the truststore is correct.

Make sure that the truststore contains the IMS Server's SSL certificate that communicates with the provisioning bridge.

Note that the truststore specified in the configuration file for the provisioning bridge is not used if the Java system property **javax.net.ssl.trustStore** and **javax.net.ssl.trustStorePassword** is set before the provisioning bridge is initialized.

To check which truststore is used, check the standard output from JVM (or console of application server). Provisioning bridge logs a message about the location of the truststore using INFO level.

*You may need to add an entry to log4j.properties like "log4j.logger.encentuate=INFO" to display this message.*

## Error Message: No security provider for the algorithm that encrypts provisioned passwords

You may need to put the provider JAR file (bcprovider-jdk**-***.jar) in the **jre\lib\ext** folder.

## Accessing ITIM, WebSphere Admin Console and IBM Tivoli Directory Server

The ITIM UI can be accessed at: http://<machine_name>/enrole/logon

The WebSphere Admin Console can be accessed at: http://<machine_name>:9090/admin/

The IBM Tivoli Directory Server can be accessed at: http://<machine_name>:9080/IDSWebApp/IDSjsp/Login.jsp

## Enabling copy and paste function in the ITIM workflow designer applet

The default Java sandbox security model does not grant clipboard permission to applet launched by browsers. Refer to the following article on steps to enable copy and paste: http://publib.boulder.ibm.com/tividd/td/ITIM/SC32-1149-02/en_US/HTML/im451_poag115.htm.

## Expiry of JCE crypto certificates when using IBM'S JRE

For WebSphere Application Server version 5.0, the IBM® JCE certificate will expire on May 18, 2006 at 21:59:19 GMT, which causes an exception.

IBM has released a patch to resolve this issue, which can be downloaded in the following location: http://www-1.ibm.com/support docview.wss?uid=swg24012195.

# Keytool

A keytool is a utility that is used to create keystores or import certificates. Refer to this appendix when you need to use a keytool and would need to refer to Sun's JRE's keytool for more guidance. This will minimize the effort of looking up the flags or parameters to provide.

When using a keytool, you can refer to Sun's Java keytool documentation in: http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html.

Below is an example of how to use the keytool:

```
keytool -import -alias ims -file <certificate_file_path> -
keystore <truststore_path> -storepass <truststore_password>
```

Before running the keytool, obtain a certificate file first:

1. Use Internet Explorer to visit https://imsserver/ where imsserver is the IMS Server's hostname
2. Click the lock icon and select **View certificates**.
3. Click the **Details** tab.
4. Click the **Copy to File...** button.
5. Click **Next**.
6. Select **Base-64 encoded X.509** format and proceed to save it to a file.

# Glossary and Abbreviations

### AccessAdmin

The management console used by individuals with the Administrator Role and/or the Helpdesk Role to administer IMS Server, and to manage users and policies.

### AccessAgent

The client software that manages the user's identity, enabling sign-on/sign-off automation and authentication management.

### AccessProfiles

Short, structured XML files that enable single sign-on/sign-off automation for applications. AccessStudio can be used to generate AccessProfiles.

### AccessStudio

The interface used to create AccessProfiles required to support end-point automation, including single sign-on, single sign-off, and customizable audit tracking.

### AD

Microsoft Active Directory

### API

Application Programming Interface

### application

An application is a software product that provides a particular function. Examples include customer relationship management systems and supply-chain management systems.

### authentication factor

The different devices, biometrics, or secrets required as credentials for validating digital identities (e.g., passwords, Encentuate USB Key, RFID, biometrics, and one-time password tokens).

### authentication service

Verifies the validity of an account; Applications authenticate against their own user store or against a corporate directory.

### CA

Certificate Authority

### CLT

Command Line Tool

### DB

Database

### DNS

Domain Name Service

### DSML

Directory Services Markup Language

### Enterprise Access Security (EAS)

A technology that enables enterprises to simplify, strengthen and track access to digital assets and physical infrastructure.

### IDI

IBM Tivoli Directory Integrator

### IMS Bridge

For extending functionalities of third party programs, allowing them to communicate with IMS Server.

### IMS Configuration Utility

A utility of the IMS Server that allows Administrators to manage lower-level configuration settings for the IMS Server.

### IMS Connectors

Add-ons to the Encentuate IMS Server that enable the IMS Server to interface with other applications as a client, extending the capability of the IMS Server.

### IMS Server

An integrated management system that provides a central point of secure access administration for an enterprise. It enables centralized management of user identities, AccessProfiles, authentication policies. provides loss management, certificate management and audit management for the enterprise.

### ITAM (IBM Tivoli Access Manager)

An integrated solution that provides a wide range of authorization and management solutions. This product can be used on various operating systems platforms such as Unix (AIX, Solaris, HP-UX), Linux, and Windows.

### ITIM (IBM Tivoli Identity Manager)

A system that integrates with Encentuate IAM to provide identity lifecycle management for application users.

### JCE

Java Cryptographic Extension

### JVM

Java Virtual Machine

### JSSE

Java Secure Socket Extension

### LDAP

Lightweight Directory Access Protocol

### M-Tech ID-Synch

An enterprise user provisioning software used by organizations.

### Mobile Active Code (MAC)

A one-time password that is randomly generated, event-based, and delivered via a secure second channel (e.g., SMS on mobile phones).

### One-Time Password (OTP)

A one-use password generated for an authentication event (e.g., password reset), sometimes communicated between the client and the server via a secure channel (e.g., mobile phones).

### policy

Governs the operation of Encentuate IAM Enterprise, comprising of two (2) main sets: machine policies (managed through Windows GPO) and IMS-managed policies (managed through AccessAdmin).

### Provisioning API

An interface that allows Encentuate IAM to integrate with user provisioning systems.

### provisioning system

A system that provides identity lifecycle management for application users in enterprises and manages their credentials.

### Radio Frequency Identification (RFID)

A wireless technology that transmits product serial numbers from tags to a scanner, without human intervention.

### single sign-on

A capability that allows a user to enter a user ID and password to access multiple applications.

### SOAP

Simple Object Access Protocol

### SSL

Secure Sockets Layer

### USB Key/ USB token

A portable and personalized device for storing user names, passwords, certificates, encryption keys, and other security credentials.

### user name (user ID)

A unique identifier that differentiates the user from all other users in the system.

### Wallet

An identity wallet that stores a user's access credentials and related information (including user IDs, passwords, certificates, encryption keys), each acting as the user's personal meta-directory.

### WebSphere Administrative Console

A graphical administrative Java application client that makes method calls to resource beans in the administrative server to access or modify a resource within the domain.